THE SECRET GUIDE

TO COMPUTERS

VOLUME 7: POPULAR LANGUAGES

tutors you in FORTRAN, PASCAL, and COBOL
compares those languages with each other, and with BASIC

turns you into an expert programmer quickly & easily— deep & fun!
call the secret phone number: a consultant will debug your program, free!

includes the newest versions: FORTRAN 77, UCSD PASCAL, COBOL 74
covers the old standards: FORTRAN IV, ANSI PASCAL, COBOL 68

news: FORTRAN's ENDIF, PASCAL's STRING, COBOL's MERGE
compares vendors: IBM, CDC, PDP, Eclipse, microcomputers

tenth edition © 1981 by Russ Walter, just $3.70
to order, use coupon on page 92

Donated to
Internet Archive by
Kay Savetz

To program a computer, you can use four popular methods. They're called BASIC, FORTRAN, PASCAL, and COBOL.

Volumes 1 and 2 of *The Secret Guide to Computers* explained BASIC. The volume you have in your hands (volume 7) explains FORTRAN, PASCAL, and COBOL. So when you finish this volume, you'll know all four methods, and you'll be a total expert.

BASIC is the easiest method. So for most applications, use BASIC. But for some applications, BASIC isn't "good enough", and you must resort to one of the other, more sophisticated, methods.

Before reading about FORTRAN, PASCAL, and COBOL, make sure you understand BASIC: read volumes 1 and 2.

Once you know BASIC, you can learn FORTRAN, PASCAL, and COBOL easily. This chart will help you translate from BASIC to the other three.

BASIC	FORTRAN	PASCAL	COBOL
CLOSE	CLOSE	*not necessary*	CLOSE
DIM X(4)	DIMENSION X(4)	X: ARRAY [1..4]	X OCCURS 4 TIMES
FOR I = 5 TO 17	DO 10 I=5,17	FOR I := 5 TO 17 DO	VARYING I FROM 5 BY 1 UNTIL I > 17.
GOSUB	CALL	*not necessary*	PERFORM
GO TO	GO TO	GOTO	GO TO
IF X=4.3	IF (X .EQ. 4.3)	IF X=4.3	IF X = 4.3
INPUT X	READ *, X	READ(X);	ACCEPT X.
INPUT#1, X	READ(1,*) X	READ(FILE1,X);	READ NUM1-FILE
NEXT I	CONTINUE	*not necessary*	*not necessary*
OPEN	OPEN	*not necessary*	OPEN
PRINT X;Y;Z	PRINT *, X,Y,Z	WRITELN(X,Y,Z);	DISPLAY X Y Z.
PRINT#1, X	WRITE(1,*) X	WRITELN(FILE1,X);	WRITE X.
REM DUMB STUFF	C DUMB STUFF	(*DUMB STUFF*)	*DUMB STUFF
RETURN	RETURN	*not necessary*	*not necessary*
STOP	STOP	GOTO 9999;	STOP RUN.
X=2.7+3.1	X=2.7+3.1	X:=2.7+3.1;	COMPUTE X = 2.7 + 3.1.

In this volume, the first chapter explains FORTRAN; the second chapter explains PASCAL; and the third chapter explains COBOL. To become a complete expert, read this volume from beginning to end. If your goal is more modest— if you want to learn just FORTRAN, or just PASCAL, or just COBOL— read just the chapter that interests you.

BASIC, FORTRAN, PASCAL, and COBOL are called *computer languages*. Though those four computer languages are the most popular, many other computer languages have been invented also. To learn more about BASIC, FORTRAN, PASCAL, COBOL, and their competitors, read the next volume (which is volume 8). To order it, use the coupon on the back page.

If you're trying to write a program in BASIC, FORTRAN, PASCAL, COBOL, or another language, and you want help, phone me at 617-266-8128. You can call 24 hours: I'm almost always in, and I sleep only lightly. I don't charge anything for the help; it's free. Many students and professional programmers have been using this free service: they read their programs to me over the telephone, and I tell them where they made mistakes. Join in: use the free service. Remember, this free consulting service is *only* via phone; do *not* expect me to correspond with you via mail.

To reduce your phone bill, call when Ma Bell's rates are low (evenings, or... better yet...nights or weekends or early mornings). Phones are cheaper than you think— you can call all the way from California to Boston for just 21¢! The phone call becomes expensive only if you dilly-dally around, instead of getting straight to the point.

To become an experienced expert, take my summer course, which gives you one-to-one tutoring in all four languages; it costs almost nothing! For details, use the coupon on the back page.

I hope you enjoy the book, and have fun becoming multilingual. One terrifying night, instead of dreaming about the opposite sex, you'll dream about the four demons: BASIC, FORTRAN, PASCAL, and COBOL. Don't get scared! Those demons will possess you: you'll start jabbering in BASIC, FORTRAN, PASCAL, and COBOL, instead of in English. No, I don't know a good exorcist. But I can fix you up with a cuddly computer.

At your service, your computer butler, Ahss

Index

FORTRAN

A=7.3 7
A=7.3+2.1 8
A=F(7.3) 36
A=SQRT(7.3) 25
ACCEPT 20, N 10
ACCEPT *, N 12
C 13
CALL YUMMY 32
CHARACTER*50 N 22
CLOSE(UNIT=3) 37
COMPLEX B 30
CONTINUE 16
DATA X/8.7/, Y/1.4/ 20
DIMENSION X(4) 18
DO 30 I=80,100 16
DOUBLE PRECISION A 29
ELSE 15
END 0
END IF 15
FORMAT (1X,A2) 21
FORMAT (1X,D22.14) 28
FORMAT (1X,E14.6) 24
FORMAT (1X,F7.2) 7
FORMAT (1X,G14.6) 7
FORMAT (1X,I3) 4
FORMAT (1X,1PE14.6) 24
FORMAT (1X,1PG14.6) 24
FORMAT (1X,'CHIP') 0
FUNCTION F(X) 36
GO TO 10 3
GO TO (10,50,20), I 16
I=-4 4
I=-4+1 8
I=IABS(-4) 26
I='UP' 21
IF (I) 20,50,90 15
IF (I .LT. 5) J=3 14
IF (I .LT. 5) THEN 15
IMPLICIT COMPLEX(C) 30
IMPLICIT DOUBLE *etc* 29
OPEN(UNIT=3, *etc* 37
PRINT 10 0
PRINT 10, N 4
PRINT *, N 12
READ 20, N 10
READ *, N 12
READ(4,20) N 37
RETURN 32
STOP *near END* 0
STOP *not near END* 3
SUBROUTINE YUMMY 32
TYPE 10 0
TYPE *, N 12
WRITE(3,10) A 37

PASCAL

A; 51
A: ARRAY *etc*; 53
A: CHAR; 52
A: INTEGER; 41
A: PACKED ARRAY *etc*; 54
A: REAL; 41
A: STRING; 54
A:='U'; 52
A:='UP'; 54
A:=2; 41
BEGIN 48
BEGIN; 38
BREAK; 42
ELSE 53
END 43
END; 42
END. 38
FOR I := 'A' TO 'E' DO 52
FOR I := 6 TO 10 DO 44
GOTO 10; 50
IF I IN [3,8,25] THEN 43
IF I<18 THEN 42
PROCEDURE INSULT; 51
PROGRAM WINE; 38
READ(X); 42
REPEAT 45
UNTIL GUESS=6; 45
VAR 41
WHILE I<1000 DO 48
WRITELN('KISS'); 38
WRITELN(SQRT(9)); 40
WRITELN(7+4); 40
10: WRITELN('KISS'); 50
{I HATE COMPUTERS} 49
(*I HATE COMPUTERS*) 49
(*$G+*) 50

COBOL

ACCEPT K. 67
ADD 7 TO A. 65
AUTHOR. 56
CLOSE POEM-FILE. 76
COMPUTE K = 53 + 4. 65
CONFIGURATION SECTION. 56
DATA DIVISION. 56
DATE-COMPILED. 90
DATE-WRITTEN. 90
DECIMAL-POINT IS COMMA. 91
DISPLAY "LIFE STINKS". 57
ENVIRONMENT DIVISION. 56
FD POEM-FILE *etc.* 76
FILE-CONTROL. 76
FILE SECTION. 76
GO TO MAIN-ROUTINE. 73
IDENTIFICATION DIVISION. 56
IF I > 5 70
INPUT-OUTPUT SECTION. 76
INSTALLATION. 90
MERGE SORT-FILE 86
MOVE "HER" TO K. 63
OBJECT-COMPUTER. 56
OPEN INPUT POEM-FILE. 77
OPEN OUTPUT POEM-FILE. 77
PERFORM HIT-ME 72
PERFORM HIT-ME. 60
PERFORM HIT-ME 6 TIMES. 60
PERFORM HIT-ME-*etc* THRU *etc.* 75
PERFORM HIT-ME UNTIL I > 90. 72
PROCEDURE DIVISION. 56
PROGRAM-ID. 56
READ POEM-FILE AT END *etc.* 79
SD SORT-FILE. 84
SECURITY. 90
SELECT POEM-FILE ASSIGN *etc.* 76
SORT SORT-FILE 84
SOURCE-COMPUTER. 56
SPECIAL-NAMES. 91
STOP RUN. 56
SUBTRACT 4 FROM B. 65
WORKING-STORAGE SECTION. 63
WRITE POEM-LINE. 78
01 C. 82
01 C PICTURE IS XXX. 63
01 C PIC XXX. 64
02 C PIC XXX. 82
02 C OCCURS *etc.* 87
02 FILLER PIC XXX. 83
*THIS IS A LOUSY PROGRAM 90

Table of contents

FORTRAN

Fun
- Simple programs 0
- Carriage controls 1
- Popular formats 2
- GO TO 3
- STOP 3

Math
- Numbers 4
- Operations 8
- Advice on variables 9

I/O
- READ 10
- Omitting formats 12

Prettier programs
- Comments 13
- Continuation 13

Logic
- IF 14
- Computed GO TO 16
- DO 16

Lists
- Subscripts 18
- Implied DO 19
- DATA 20
- String variables 21

Fancy formats
- Double apostrophe 23
- T format 23
- Blank records 23
- Repeated formats 24
- E format 24
- P format 24

Functions
- Square root 25
- FLOAT 26
- Random numbers 26
- Maxima and minima 26
- Absolute value 26
- Remainder 26
- Trigonometry 27
- Calculus 27

Exotic features
- DOUBLE PRECIS'N 28
- COMPLEX 30
- Subroutines 32
- Your own functions 36
- Files 37

PASCAL

Fun
- Simple programs 38
- Multiple writing 39
- Math 39
- Variables 41
- READ 42
- IF 42

Loops
- FOR 44
- REPEAT...UNTIL 45
- WHILE 47

Exotic features
- Alternatives 48
- Comments 49
- GOTO 50
- PROCEDURE 51
- CHAR 52
- Subscripts 53
- Writing strings 54
- Reading strings 55

COBOL

Fun
- Simple programs 56
- Abridgments 59
- Fancy displays 59
- PERFORM 60

Variables
- String variables 63
- Numeric variables 65
- Decimals 66
- Negatives 66
- ACCEPT 67
- Editing 68

Logic
- IF 70
- UNTIL 72
- GO TO 73
- THRU 75

Structures
- Files 76
- Group items 82
- SORT 84
- MERGE 86
- Subscripts 87

Exotic features
- IDENTIFICATION 90
- Comments 90
- DECIMAL-POINT 91

Note to readers of the previous edition: you'll particularly enjoy pages 15, 38-55, and 86-89, which contain new material. You'll also get a kick out of pages 56-58, 60-62, and 76-77, which use new teaching techniques.

FORTRAN

Volumes 1 and 2 explained BASIC. Now you'll learn a different computer language, called FORTRAN. Most maxicomputers and minicomputers understand both BASIC and FORTRAN. Some ideas are easier to express in BASIC, others are easier in FORTRAN. Most scientists and engineers use FORTRAN, not BASIC. The language universities most commonly teach is FORTRAN.

IBM invented the first version of FORTRAN in 1957. Then came improvements, called FORTRAN II, FORTRAN III, and FORTRAN IV. The newest version is called FORTRAN 77, because it was invented in 1977. Today, some computers use FORTRAN 77; others still use FORTRAN IV, or a slightly souped-up version of it (called FORTRAN IV-EXTENDED or FORTRAN V or WATFOR or WATFIV or FORTRAN 10). But have no fear: the features of FORTRAN discussed in this chapter work on practically all computers.

Fun

SIMPLE PROGRAMS

Here's a FORTRAN program:

```
      PRINT 10
10    FORMAT (1X,'CHIPMUNKS ARE CHUBBY')
      PRINT 20
20    FORMAT (1X,'GOLDFISH GIGGLE')
      PRINT 10
      END
```

The top line says to print what's in line 10, so the computer will print CHIPMUNKS ARE CHUBBY. The next line says to print what's in line 20, so the computer will print GOLDFISH GIGGLE. The next line says to print what's in line 10, so the computer will print CHIPMUNKS ARE CHUBBY again. Altogether, the program makes the computer print:

```
CHIPMUNKS ARE CHUBBY
GOLDFISH GIGGLE
CHIPMUNKS ARE CHUBBY
```

Notice:

Each line of the program is indented 6 spaces, so it begins in the 7th position.
Each FORMAT begins with 1X.
Each string is enclosed in apostrophes.

You must number every line that's referred to. For example, you must number the FORMAT lines, since the PRINT lines refer to them. You don't have to number the PRINT lines.

The bottom line of every FORTRAN program must be END.

That program works on most computers. On PDP computers, say TYPE instead of PRINT. On CDC computers using TS FORTRAN, replace each apostrophe by an asterisk. On IBM computers, put STOP above END, so the bottom two lines of your program are:

```
      STOP
      END
```

Ask the people in your computer center how to feed a FORTRAN program into the computer. On some computers, you must put an edit number in front of each line:

```
00100          PRINT 10
00110    10    FORMAT (1X,'CHIPMUNKS ARE CHUBBY')
00120          PRINT 20
00130    20    FORMAT (1X,'GOLDFISH GIGGLE')
00140          PRINT 10
00150          END
```

Some computers don't require you to indent each line. To indent quickly on PDP computers, hold down the CONTROL key, while you type the letter I.

Each line of your program must be short: no more than 72 characters, including the spaces at the beginning of the line.

CARRIAGE CONTROLS

The 1X at the beginning of each FORMAT is called the *carriage control*. It means: print the format normally. For weirder printing, replace the 1X by '0' or '1' or '+':

```
      PRINT 10
10    FORMAT (1X,'NIFTY')
      PRINT 20
20    FORMAT ('0','SAL')
      END
```

In line 10, the 1X makes the computer print NIFTY normally. In line 20, the zero-in-apostrophes makes the computer leave a blank line, then print SAL. The computer will print:

```
NIFTY

SAL
```

Suppose you change the carriage control to '1':

```
      PRINT 10
10    FORMAT (1X,'NIFTY')
      PRINT 20
20    FORMAT ('1','SAL')
      END
```

If your terminal uses paper, the '1' makes the computer print SAL on a new page. If your terminal uses a screen instead of paper, the '1' makes the computer erase the screen before printing SAL.

Suppose you change the carriage control to '+':

```
      PRINT 10
10    FORMAT (1X,'NIFTY')
      PRINT 20
20    FORMAT ('+','SAL')
      END
```

If your terminal uses paper, the '+' makes the computer print SAL in the same place as NIFTY, like this:

```
SALTY
```

If your terminal uses a screen instead, the computer will print NIFTY, but then the NIF will suddenly disappear, and you'll see SALTY.

To print the symbol ϴ, print 0 in the same place as -. To print the symbol ≠, print = in the same place as /. To print ϴ≠A, print 0=A in the same place as -/. If your terminal uses paper, this program prints ϴ≠A:

```
      PRINT 10
10    FORMAT (1X,'0=A')
      PRINT 20
20    FORMAT ('+','-/')
      END
```

If you're using a Hazeltine terminal or CDC TS FORTRAN, the carriage controls won't work.

POPULAR FORMATS

Examine this program:

```
      PRINT 10
10    FORMAT (1X,'JOHN','NY')
      END
```

The FORMAT consists of three *fields*: the first is the carriage control 1X, the second is 'JOHN', and the third is 'NY'. The computer will print JOHN and NY on the same line:

```
JOHNNY
```

Another example:

```
      PRINT 10
10    FORMAT (1X,'EAT A',8X,'MEATBALL')
      END
```

The computer will print EAT A, then 8 blank spaces, then MEATBALL:

```
EAT A        MEATBALL
```

Here's another program for meatball lovers:

```
      PRINT 10
10    FORMAT (1X,'EAT A',1X,'MEATBALL')
```

The computer will print EAT A, then 1 blank space, then MEATBALL:

```
EAT A MEATBALL
```

If you say X instead of 1X, the computer will gripe.

This program quotes Julius Caesar:

```
      PRINT 10
10    FORMAT (1X,'I CAME')
      PRINT 20
20    FORMAT (1X,'I SAW')
      PRINT 30
30    FORMAT (1X,'I CONQUERED')
      END
```

The computer will print:

```
I CAME
I SAW
I CONQUERED
```

This program does the same thing:

```
      PRINT 10
10    FORMAT (1X,'I CAME'/1X,'I SAW'/1X,'I CONQUERED')
      END
```

Line 10 consists of 3 *records*: the first is 1X,'I CAME'; the second is 1X,'I SAW'; the third is 1X,'I CONQUERED'. Each record begins with a carriage control; the records are separated by slashes. The computer will print each record on a separate line.

GO TO

You can say GO TO:

```
10    PRINT 20
20    FORMAT (1X,'CAT')
      PRINT 30
30    FORMAT (1X,'DOG')
      GO TO 10
      END
```

The top line says to print what's in line 20, so the computer will print:

```
CAT
```

The next line says to print what's in line 30, so the computer will print:

```
DOG
```

The next line makes the computer go back to line 10. The computer will print CAT again, then DOG again, then jump back to line 10 again....The computer will try to print the words CAT and DOG again and again, forever.

STOP

The computer understands the word STOP:

```
      PRINT 10
10    FORMAT (1X,'BUBBLE GUM')
      PRINT 20
20    FORMAT (1X,'SHAKESPEARE')
      STOP
      PRINT 30
30    FORMAT (1X,'DADA')
      END
```

The top line says to print what's in line 10, so the computer will print BUBBLE GUM. The next line says to print what's in line 20, so the computer will print SHAKESPEARE. The next line says STOP, so the computer will stop. It will never print DADA.

Math

NUMBERS

FORTRAN distinguishes between *integers* and *real numbers*. Here's how FORTRAN defines them....

An *integer* contains no decimal point and no exponent.

Integer	Not an integer	Comment
-27	-27.0	An integer contains no decimal point.
50000	5E4	An integer contains no exponent.

A *real number* contains either a decimal point or the letter E.

Real number	Not a real number
-27.0	-27
2.35E8	235000000
5E4	50000

The largest permissible integer is different from the largest permissible real:

Computer	Largest integer	Largest real	Tiniest real
PDP-11 using FORTRAN IV	32767	1.7E38	2.9E-39
PDP-11 using FORTRAN IV-PLUS	2147483647	1.7E38	2.9E-39
PDP-10 or Honeywell	34359738367	1.7E38	2.9E-39
IBM	2147483647	7.2E75	5.4E-79
CDC	281474976710655	2.5E322	3.1E-294

In BASIC, X can stand for a number, such as 3.7. The same is true in FORTRAN. A variable can be a letter (such as X) or a letter-followed-by-a-combination-of-letters-and-digits (such as FUN4U2). The variable must be short: no more than 6 characters. AVERAGE is too long: say AVERAG instead. If the variable begins with I, J, K, L, M, or N, it stands for an integer. If it begins with some other letter, it stands for a real number.

Example:

```
      JUNKY=-47
      PRINT 10, JUNKY
10    FORMAT (1X,I3)
      END
```

Since JUNKY begins with J, it stands for an integer. The first line says JUNKY stands for the integer -47. The second line says to print JUNKY, using line 10. Line 10 explains how to print JUNKY. The I3 means: print it as an Integer having 3 characters. The computer will print:

```
-47
```

If you change the I3 to I4, the computer will print JUNKY as an Integer having 4 characters. To print a total of 4 characters, the computer will print a blank space in front of -47, like this:

```
 -47
```

If you change to I5, the computer will print JUNKY as an Integer having 5 characters, by printing two blank spaces in front of -47:

```
  -47
```

If you change to I2, the computer will try to print JUNKY as an integer having 2 characters. But it's impossible to express -47 by using only 2 characters. The computer will obey the format and print 2 characters, but will make them asterisks:

```
**
```

Another example:

```
      NUM=31.9
      PRINT 10, NUM
10    FORMAT (1X,I4)
      END
```

Since NUM begins with N, it stands for an integer. The program's top line tries to make NUM stand for 31.9; but that's impossible, since 31.9 isn't an integer. The computer will omit the .9 and make NUM stand for the integer 31. The computer will print 31, using an I4 format:

```
 31
```

Example:

```
      JOE=-5.8
      PRINT 10, JOE
10    FORMAT (1X,I4)
      END
```

The computer will set JOE equal to the integer -5 and print it:

```
-5
```

Example:

```
      JAIL=74
      KRIMNL=829
      PRINT 10, JAIL,KRIMNL
10    FORMAT (1X,I2,I3)
      END
```

Since JAIL begins with J, and KRIMNL begins with K, they're both integers. The computer will print JAIL and KRIMNL, using the format in line 10. The format says to print a 2-character integer, then a 3-character integer. The computer will print:

```
74829
```

If you change the format to (1X,I2,4X,I3), the computer will print a 2-character integer, then 4 blanks, then a 3-character integer:

```
74    829
```

If you change the format to—

```
10    FORMAT (1X,'JAIL NUMBER',1X,I2,1X,'CONTAINS CRIMINAL',1X,I3)
```

the computer will print JAIL NUMBER, then a blank, then a 2-character integer, then a blank, then CONTAINS CRIMINAL, then a blank, then a 3-character integer:

```
JAIL NUMBER 74 CONTAINS CRIMINAL 829
```

Example:

```
      J=43
      K=75
      L=96
      M=81
      N=24
      PRINT 10, J,K,L,M,N
10    FORMAT (1X,I2,I2,I2,I2,I2)
      END
```

The computer will print 43, then 75, then 96, then 81, then 24:

```
4375968124
```

You can write that format more briefly:

```
10    FORMAT (1X,5I2)
```

If you change the format to (1X,5I3), the computer will print each integer as 3 characters— a blank followed by two digits:

```
 43 75 96 81 24
```

If you change the format to (1X,I3), the computer will print only one integer per line:

```
 43
 75
 96
 81
 24
```

If you change the format to (1X,2I3), the computer will print 2 integers per line:

```
 43 75
 96 81
 24
```

If you change the format to (1X,'GOSH',I3,1X,'SUPERB',I3,1X,'JEEPERS'), the computer will print 2 integers per line:

```
GOSH 43 SUPERB 75 JEEPERS
GOSH 96 SUPERB 81 JEEPERS
GOSH 24 SUPERB
```

To be safe, use I14 format for integers. On most computers, I14 handles even the largest integers, and prints blank spaces between them.

Integers are also called *fixed-point* numbers. Real numbers are called *floating-point* numbers.

The I format is only for integers. For real numbers, use F or G. The F format is easy to understand:

```
      RADIUS=-586.39
      PRINT 10, RADIUS
10    FORMAT (1X,F7.2)
      END
```

Since RADIUS doesn't begin with I, J, K, L, M, or N, it stands for a real number. The first line says RADIUS stands for the real number -586.39. The second line says to print RADIUS, using the format in line 10. The F7.2 means: print it as a floating-point number having 7 characters, 2 of them after the decimal point. The computer will print:

```
-586.39
```

If you change the F7.2 to a different format, the following chart shows what happens; in the chart, each ▮ represents a blank space:

Format	What the computer will print	Comment
F8.2	▮-586.39	To print 8 characters instead of 7, it prints a blank space at the beginning.
F8.3	-586.390	To print 3 characters after the decimal point instead of 2, it prints a zero at the end.
F8.1	▮▮-586.4	To print 1 character after the decimal point instead of 2, it rounds the .39 to .4.
F8.4	********	To print 4 characters after the decimal point, the computer would have to print -586.3900; since that requires more than 8 characters, the computer complains by printing asterisks.

To print a real number, the safest format is G14.6. The G14.6 handles *all* real numbers— even the largest— so it is more flexible than F. The G14.6 prints 14 characters altogether, 6 of which are significant digits. Here are examples of numbers printed in G14.6 format:

```
▮-0.283941E-29
▮▮0.293027▮▮▮▮
▮▮▮5.34523▮▮▮▮
▮▮▮39.4539▮▮▮▮
▮▮▮47802.3▮▮▮▮
▮▮▮986327.▮▮▮▮
▮▮0.288341E+24
```

Example:

```
      PRUNES=17
      PRINT 10, PRUNES
10    FORMAT (1X,G14.6)
      END
```

Since PRUNES doesn't begin with I, J, K, L, M, or N, it stands for a real number. When the computer encounters the first line of the program, it will set PRUNES equal to the real number 17.0. It will print:

```
17.0000
```

The program will run faster if you change the top line to this:

```
      PRUNES=17.0
```

OPERATIONS

For addition, subtraction, multiplication, and division, FORTRAN uses the same symbols as BASIC.

```
      N=2*(3+1)
      S=7.3+2.1
      PRINT 10, N,S
10    FORMAT (1X,I14,G14.6)
      END
```

Since N is 8, and S is 9.4, the computer will print:

```
             8   9.40000
```

For exponents, FORTRAN uses a double star:

```
      J=7**2
      P=.5**3
      PRINT 10, J,P
10    FORMAT (1X,I14,G14.6)
      END
```

Since J is 7^2 (which is 49), and P is $.5^3$ (which is .125), the computer will print:

```
            49   0.125000
```

For negative exponents, you need parentheses. You must say 6.1**(-2), not 6.1**-2.

When you combine integers, the answer is an integer:

2+3 is 5
8-8 is 0
2*4 is 8
399/100 is 3 (not 3.99)
11/4 is 2 (not 2.75)
3/4 is 0 (not 0.75)
10**(-2) is 0 (not 0.01)

When you combine real numbers, the answer is real:

4.1+2.9 is 7.0 (not 7)
8.0-8.0 is 0.0 (not 0)
399.0/100.0 is 3.99
11.0/4.0 is 2.75
3.0/4.0 is .75
10.0**(-2.0) is .01

When you combine an integer with a real number, the answer is real:

3+2.0 is 5.0
399/100.0 is 3.99
11/4.0 is 2.75
3/4.0 is .75
10.0**(-2) is .01

Compare these:

7/10*10 is 0 (because 7/10 is 0)
7/10*10.0 is 0.0 (because 0*10.0 is 0.0)
7/10.0*10 is 7.0 (because 7/10.0 is .7)

Example:

```
      JERK=20.9+30.9
      PRINT 10, JERK
10    FORMAT (1X,I14)
      END
```

Since JERK begins with J, it stands for an integer. Since 20.9+30.9 is 51.8, JERK stands for the integer 51. The computer will print:

```
          51
```

Another example:

```
      APPLE=37/10
      PRINT 10, APPLE
10    FORMAT (1X,G14.6)
      END
```

Since APPLE begins with A, it stands for a real number. Since 37/10 is 3, APPLE stands for the real number 3.0. The computer will print:

```
3.00000
```

If you commit one of these crimes, the computer will work slowly....

little crime: use a real number
medium crime: mix integers with reals
big crime: use a real exponent

For example, 2.0+2.0 is slower than 2+2, because 2.0+2.0 is a little crime. 2.1+7 is slower than 2.1+7.0, because 2.1+7 is a medium crime, whereas 2.1+7.0 is only a little crime. 5.1**2.0 is slower than 5.1**2, because 5.1**2.0 is a big crime, whereas 5.1**2 is only a medium crime. X=0 is slower than X=0.0, because X=0 is a medium crime, wnereas X=0.0 is only a little crime.

5**3.1 is a gigantic crime, since it is a medium crime and a big crime simultaneously. Because the crime is so gigantic, some computers refuse to do it. Say 5.0**3.1 instead.

ADVICE ABOUT VARIABLES

FORTRAN, like BASIC, distinguishes variables, constants, and expressions:

X is a variable
2.7 is not a variable; it's a *numeric constant*
'LOVE' is not a variable; it's a *string constant*
X+Y is not a variable; it's an *expression*

In a PRINT statement, some computers allow only variables....

allowed: PRINT 10, X
not allowed: PRINT 10, 2.7 *instead, say X=2.7 and PRINT 10, X*
not allowed: PRINT 10, 'LOVE' *instead, say PRINT 10 and 10 FORMAT (1X, 'LOVE')*
not allowed: PRINT 10, X+Y *instead, say Z=X+Y and PRINT 10, Z*

Other computers are more generous and allow anything. Find out about yours.

To help other humans understand your program, use long variable names throughout your program. Say RADIUS, not R; say AREA, not A; say VOLUME, not V; say SUM, not S; say TOTAL, not T. Because FORTRAN's variables are restricted to six characters, you might have to omit the last few syllables (*revolutions* becomes REVOLU) or the last few vowels (RVLTNS).

If you want a variable to be real, but its English name begins with I, J, K, L, M, or N, begin its FORTRAN name with an A (*mass* becomes AMAS; *length* becomes ALENGT or ALNGTH). If you want a variable to be an integer, but its English name doesn't begin with I, J, K, L, M, or N, begin its FORTRAN name with an I (*population* become IPOPUL) or misspell it (*count* becomes KOUNT) or choose a synonym (instead of *position*, say *location*, which is LOCATN).

I/O

READ

The computer can read.

```
      PRINT 10
10    FORMAT (1X,'TYPE SOME DIGITS')
      READ 20, N
20    FORMAT (I4)
      PRINT 30, N
30    FORMAT (1X,I4)
      END
```

When you run the program, the first pair of lines make the computer print:

```
TYPE SOME DIGITS
```

The word READ makes the computer wait for you to type something; it's like the BASIC word INPUT. The computer will wait for you to type the value of N, but line 20's FORMAT makes the computer read just the first 4 characters. For example, suppose you type:

```
-75198622
```

The computer will read just the first 4 characters, which are -751; it will ignore the 98622; so N will be -751. Line 30's FORMAT makes the computer print:

```
-751
```

Altogether, the run looks like this:

```
The computer says→→→→→TYPE SOME DIGITS
You say→→→→→→→→→→→→→→→→-75198622
The computer replies→→→-751
```

Line 30's FORMAT contains a carriage control 1X, but line 20's FORMAT omits the carriage control. Put a carriage control in formats that PRINT, but not in formats that READ.

On PDP computers, say ACCEPT instead of READ.

If you input a blank space, the computer treats it as a zero. For example, suppose you input:

```
-3 28219
```

Because of the I4 format, the computer will read just the first 4 characters, which are -3 2; the blank space between the 3 and the 2 is treated as a zero, so N will be -302.

Suppose you input:

```
57
```

Because of the I4 format, the computer will read the 5, the 7, and two blanks. Since the blanks are treated as zeros, N will be 5700.

Suppose you input:

```
■■■9527
```

Because of the I4 format, the computer will read the three beginning blanks and the 9. Since the blanks are treated as zeros, N will be 0009, which is 9. Line 30 makes the computer print:

```
 9
```

Suppose you write a program containing these lines:

```
      READ 20, L,M,N
20    FORMAT (I3,I4,2X,I2)
```

When you run that program, suppose you input:

```
58194138972824
```

The I3 format makes the first 3 characters (581) be L; the I4 format makes the next 4 characters (9413) be M; the 2X format makes the next 2 characters (89) be skipped over; the I2 format makes the next two characters (72) be N; the remaining characters (824) are ignored. So the line is split like this:

variables, in the READ statement →	L	M		N	
fields, in the FORMAT statement →	I3	I4	2X	I2	
line you input →	581	9413	89	72	824

Suppose you write a program containing these lines:

```
      READ 20, J,K,L,M,N
20    FORMAT (2I3)
```

The format says to read two 3-character integers on each line. Suppose you input:

```
78345692
85431684
46185327
```

J will be 783, and K will be 456. L will be 854, and M will be 316. N will be 461.

That's how to input integers. Here's how to input real numbers instead....

```
      PRINT 10
10    FORMAT (1X,'TYPE SOME DIGITS')
      READ 20, P
20    FORMAT (F6.2)
      PRINT 30, P
30    FORMAT (1X,G14.6)
      END
```

The F6.2 format means: read 6 characters; if they don't contain the decimal point, insert it before the last 2 digits. For example, suppose you input:

```
327514968
```

The computer reads the first 6 characters (327514). Since they don't contain the decimal point, the computer inserts it before the last 2 digits, so P is 3275.14. Line 30 prints:

```
 3275.14
```

For that program, suppose you input:

```
7.5423967
```

The computer reads the first 6 characters (7.5423). Since they already contain the decimal point, F is 7.5423. Line 30 prints:

```
 7.54230
```

Suppose you input:

```
497E3
```

The computer reads 6 characters (497E3, followed by a blank). Since blanks are treated as zeros, the computer gets 497E30. Since 497E30 doesn't contain the decimal point, the computer inserts it before the last 2 digits, so P is 4.97E30. Line 30 prints:

```
 0.497000E+31
```

OMITTING FORMATS

Most computers let you omit numeric formats. This program works on most modern computers (such as computers having FORTRAN 77, PDP-20 computers, PDP-10 computers using FORTRAN 10, PDP-11 computers using FORTRAN IV-PLUS, CDC computers using FORTRAN IV-EXTENDED, and IBM computers using FORTRAN H-EXTENDED):

```
      PRINT 10
10    FORMAT (1X,'TYPE TWO INTEGERS')
      READ *, M,N
      ISUM=M+N
      PRINT *, ISUM
      END
```

On PDP computers, say TYPE instead of PRINT, and say ACCEPT instead of READ.

The word READ is followed by an asterisk, instead of a FORMAT number. The last PRINT is followed by an asterisk also. The asterisk makes the computer invent its own FORMAT. To make the program add 241 and 82976, input the numbers, separated by a comma:

```
241,82976
```

The computer will notice the comma's location and automatically use an I3 format for 241, a 1X format to skip over the comma, and an I5 format for 82976. To print ISUM, the computer will use a safe format, such as I14 or I15.

By omitting formats, you gain two advantages:

✓ You can write FORTRAN programs more quickly.

✓ The person who inputs doesn't have to worry about whether his spacing matches the format. The computer invents a format that matches his input.

On CDC computers using TS FORTRAN, and on Honeywell computers, omit the asterisk after READ and PRINT:

```
      PRINT 10
10    FORMAT (1X,'TYPE THE NUMBERS')
      READ, M,N
      ISUM=M+N
      PRINT, ISUM
      END
```

On PDP-10 computers using F40 FORTRAN, and on PDP-11 computers using regular FORTRAN IV, you need FORMATs, but omit the number in the I format:

```
      TYPE 10
10    FORMAT (1X,'TYPE THE NUMBERS')
      ACCEPT 20, M,N
20    FORMAT (1X,2I)
      ISUM=M+N
      TYPE 30, ISUM
30    FORMAT (1X,I)
      END
```

You can use similar short cuts for real numbers.

Prettier programs

COMMENTS

If you type C instead of a line number, the computer will ignore the line.

```
      N=50+13
C  I HATE COMPUTERS!
      PRINT 10, N
10    FORMAT (1X,I14)
      END
```

The computer will ignore the Comment. The computer will print 63.

The C in FORTRAN is like the REMARK in BASIC: use it to document your program.

CONTINUATION

Some computers look at only the first 72 characters of each line of your program: if your line contains more than 72 characters, it won't work.

If you want to type a long statement, type just the first 72 characters. On the line below, type the remaining characters, beginning in the 7th print position; and in the 6th print position type a 6:

```
      PRINT 10
10    FORMAT (1X,'I LIKE ROSES IN MY TEA.'/1X,'THEY MAKE IT GLOW RED, LI
     6KE HOT BLOOD.')
      END
```

The computer will print:

```
I LIKE ROSES IN MY TEA.
THEY MAKE IT GLOW RED, LIKE HOT BLOOD.
```

A line that has a 6 in column 6 is called a *continuation line*, because it's a continuation of the line above it.

(I suggest you put a 6 in column 6. But some programmers put a 1 in column 6, or a 9 in column 6, or a + in column 6, or a * in column 6, or a $ in column 6. In fact, you can put *any character* in column 6, except a zero. Choose your favorite character; the computer doesn't care.)

On PDP computers, put the 6 immediately after a controlled I, instead of in column 6:

```
10    FORMAT (1X,'I LIKE ROSES IN MY TEA.'/1X,'THEY MAKE IT GLOW RED, LI
      6KE HOT BLOOD.')
```

(I suggest you put a 6 after the controlled I. But you can put *any non-zero digit*, instead of a 6.)

On CDC computers using TS FORTRAN, type a + instead of a 6, and put it immediately after the edit number, with no intervening spaces:

```
00120 10 FORMAT (1X,'I LIKE ROSES IN MY TEA.'/1X,'THEY MAKE IT GLOW RED, LI
00130+KE HOT BLOOD.')
```

Logic

IF

FORTRAN allows the word IF:

Statement	Meaning
`IF (I .LT. 5) J=3`	If I is Less Than 5, let J=3.
`IF (I .LT. 5) GO TO 80`	If I is Less Than 5, go to line 80.
`IF (I .LT. 5) STOP`	If I is Less Than 5, stop.
`IF (I .LT. 5) PRINT 10, J`	If I is Less Than 5, print J by using line 10's FORMAT.

The condition must be enclosed in parentheses. LT must be enclosed in periods.
FORTRAN uses these clauses:

Clause	Meaning
`IF (I .LT. 5)`	if I is Less Than 5
`IF (I .GT. 5)`	if I is Greater Than 5
`IF (I .LE. 5)`	if I is Less than or Equal to 5
`IF (I .GE. 5)`	if I is Greater than or Equal to 5
`IF (I .EQ. 5)`	if I is EQual to 5
`IF (I .NE. 5)`	if I is Not Equal to 5

You can use the words AND and OR:

Idea	How to say it in FORTRAN
if I is 2 or 9 or 13	`IF (I .EQ. 2 .OR. I .EQ. 9 .OR. I .EQ. 13)`
if I is an integer from 1 to 100	`IF (I .GE. 1 .AND. I .LE. 100)`
if X<Y<Z	`IF (X .LT. Y .AND. Y .LT. Z)`
if A is less than both B and C	`IF (A .LT. B .AND. A .LT. C)`
if X is negative or between 5 and 9	`IF (X .LT. 0.0 .OR. X .GE. 5.0 .AND. X .LE. 9.0)`

Take this test: cover the column that says "How to say it in FORTRAN". Try to translate each "Idea" into FORTRAN, then check your answers. If one of your answers is shorter than the correct answer, take the test again! For example, the following answer to the first idea is *wrong*:

```
IF (I .EQ. 2 .OR. 9 .OR. 13)
```

FORTRAN 77 lets you say "END IF". For example, here's how FORTRAN 77 lets you say, "If I is greater than 5, let J be 80 and let K be 90":

```
IF (I .GT. 5) THEN
   J=80
   K=90
END IF
```

Here's how FORTRAN 77 lets you say, "If I is greater than 5, let J be 80 and let K be 90; but if I is *not* greater than 5, let J be 30 and let K be 50":

```
IF (I .GT. 5) THEN
   J=80
   K=90
ELSE
   J=30
   K=50
END IF
```

Here's how FORTRAN 77 lets you say, "If I is greater than 5, let J be 80 and let K be 90; if I is *not* greater than 5, but I is greater than 2, let J be 81 and let K be 92; if I is not greater than 2, let J be 30 and let K be 50":

```
IF (I .GT. 5) THEN
   J=80
   K=90
ELSE IF (I .GT. 2) THEN
   J=81
   K=92
ELSE
   J=30
   K=50
END IF
```

Warning: to say "END IF", you must get FORTRAN 77. If you use FORTRAN IV instead, "END IF" doesn't work. I recommend that you get FORTRAN 77.

Here's a different kind of IF statement:

```
IF (X) 20,50,90
```

It means:

if X is a negative number, go to line 20;
if X is zero, go to line 50;
if X is a positive number, go to line 90.

That kind of IF statement is called a *three-way IF*, or an *arithmetic IF*. (To pronounce "arithmetic", put the accent on met.) The other kind of IF is called a *logical IF*.

COMPUTED GO TO

In your program, you can say:

```
GO TO (80,100,20,350), I
```

That means: go to either 80, 100, 20, or 350, depending on what I is. More specifically, it means:

go to line 80, if I is 1;
go to line 100, if I is 2;
go to line 20, if I is 3;
go to line 350, if I is 4;
proceed to the line underneath, if I is a different integer.

That FORTRAN statement is called a *computed GO TO*. It resembles this BASIC statement:

```
30 ON I GO TO 80,100,20,350
```

DO

This program prints the square of every number from 80 to 100, and then prints GET LOST:

BASIC

```
10 FOR I = 80 TO 100
20    PRINT I↑2
30 NEXT I
40 PRINT "GET LOST"
```

FORTRAN

```
      DO 30 I=80,100
         J=I**2
         PRINT 20, J
20       FORMAT (1X,I14)
30    CONTINUE
      PRINT 40
40    FORMAT (1X,'GET LOST')
      END
```

If you compare the BASIC with the FORTRAN, you'll notice FORTRAN uses the word DO instead of FOR, uses a comma instead of TO, and uses CONTINUE instead of NEXT. The statement *DO 30 I=5,9* means: DO every line up through line 30, repeatedly, as I goes from 5 to 9. In BASIC, programmers indent every line between FOR and NEXT; in FORTRAN, programmers indent every line between DO and CONTINUE. In BASIC, the indented lines are called a *FOR...NEXT loop*; in FORTRAN, they're called a *DO loop*.

If you want the computer to print the square of every fifth number from 80 to 100, change the top line of the program:

BASIC

```
10 FOR I = 80 TO 100 STEP 5
```

FORTRAN

```
DO 30 I=80,100,5
```

In a DO statement, some computers allow only positive integer variables and constants:

Not allowed	Why	What to say instead
`DO 10 X=1.0,5.0`	reals are not allowed	`DO 10 I=1,5`
`DO 10 X=17.3,98.5`	reals are not allowed	`DO 10 I=173,985` `X=I/10.0`
`DO 10 I=0,5`	0 is not positive	`DO 10 J=1,6` `I=J-1`
`DO 10 I=-3,5`	-3 is not positive	`DO 10 J=1,9` `I=J-4`
`DO 10 I=100,7,-1`	-1 is not positive	`DO 10 J=7,100` `I=107-J`
`DO 10 I=5,J+K`	+ is not allowed	`L=J+K` `DO 10 I=5,L`

Other computers are more generous and allow anything. Find out about yours.

If you say—

```
DO 10 I=1,N
```

the computer will do up through line 10, N times. For example, if N is 73, the computer will do up through line 10, 73 times. If N is 2, the computer will do up through line 10, twice. If N is 1, the computer will do up through line 10, once.

What happens in N is less than 1? The answer depends on which version of FORTRAN you're using. If you're using FORTRAN 77, the computer will skip the loop, and proceed to the line below line 10. But if you're using FORTRAN IV, the computer will do the loop once, as if N were 1. FORTRAN 77 makes more sense, but at the moment most computers are still using FORTRAN IV.

In the middle of a DO loop, don't change the value of the index. For example, if your DO loop begins with—

```
DO 10 I=1,100
```

don't insert this line in the middle of your loop:

```
I=14
```

It will confuse the computer.

Lists

SUBSCRIPTS

Like BASIC, FORTRAN permits subscripts:

```
      DIMENSION X(4)
      X(1)=.21
      X(2)=.3
      X(3)=1.08
      X(4)=5.0
      SUM=X(1)+X(2)+X(3)+X(4)
      PRINT 10, X,SUM
10    FORMAT (1X,G14.6)
      END
```

X will be a list of 4 numbers, called X(1), X(2), X(3), and X(4). Since X doesn't begin with I, J, K, L, M, or N the 4 numbers will be real.

The computer will print the list and the SUM, like this:

```
 0.210000
 0.300000
  1.08000
  5.00000
  6.59000
```

If you change the format to (1X,5G14.6), the computer will print all 5 numbers on the same line:

```
 0.210000      0.300000      1.08000      5.00000      6.59000
```

You must say DIMENSION if your program uses subscripts, even if the subscripts are small. Say DIMENSION at the very top of the program. Make sure you say DIMENSION, not DIM or DIMENSIONS. The computer assumes all subscripts will be positive, so don't say X(0). If a subscript is zero or negative or larger than the DIMENSION statement says, the computer might not notice your error, and will print wrong answers without warning you.

If your program begins like this—

```
      DIMENSION A(6)
      READ 10, A
```

the computer will begin by reading 6 real numbers, which will become A(1), A(2), A(3), A(4), A(5), and A(6).

If you want T to be a table of numbers, and you want T to have 4 rows and 2 columns, begin your program by saying:

```
      DIMENSION T(4,2)
```

To print T, say:

```
      PRINT 10, T(1,1), T(1,2)
      PRINT 10, T(2,1), T(2,2)
      PRINT 10, T(3,1), T(3,2)
      PRINT 10, T(4,1), T(4,2)
10    FORMAT (1X,2G14.6)
```

If you say—

```
      PRINT 10, T
```

the computer will print the entire table T, but in an undesirable order: it will print T(1,1), T(2,1), T(3,1), and T(4,1), then T(1,2), T(2,2), T(3,2), and T(4,2). Similarly, "READ 5, T" makes the computer read T in an undesirable order.

IMPLIED DO

This statement—

```
PRINT 10, X(3),X(4),X(5),X(6),X(7)
```

can be written more briefly, like this:

```
PRINT 10, (X(I), I=3,7)
```

It means: using line 10's format, print the value of X(I), for I = 3 to 7. The construction (X(I), I=3,7) is called an *implied DO loop*. Notice the parenthesis at the beginning, the parenthesis at the end, and the commas. Here are other examples of implied DO loops:

Implied DO loop	Meaning
(X(I), I=100,120,5)	X(100),X(105),X(110),X(115),X(120)
(X(I),Y(I), I=3,7)	X(3),Y(3), X(4),Y(4), X(5),Y(5), X(6),Y(6), X(7),Y(7)

Here's a calendar:

```
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

This program prints it:

```
      PRINT 10, (I, I=1,31)
10    FORMAT (1X,I2,1X,I2,1X,I2,1X,I2,1X,I2,1X,I2,1X,I2)
      END
```

The program's top statement says to print every value of I, for I = 1 to 31. The FORMAT says to print 7 integers on each line, and separate the integers by spaces. Since 1X followed by I2 is about the same as I3, you can write the FORMAT more briefly:

```
10    FORMAT (I3,I3,I3,I3,I3,I3,I3)
```

Even more briefly:

```
10    FORMAT (7I3)
```

If T is a table having 3 rows and 5 columns, these lines will print it in the correct order:

```
      PRINT 20, T(1,1), T(1,2), T(1,3), T(1,4), T(1,5)
      PRINT 20, T(2,1), T(2,2), T(2,3), T(2,4), T(2,5)
      PRINT 20, T(3,1), T(3,2), T(3,3), T(3,4), T(3,5)
20    FORMAT (1X,5G14.6)
```

Tbis short cut does the same thing:

```
      PRINT 20, (T(1,J), J=1,5)
      PRINT 20, (T(2,J), J=1,5)
      PRINT 20, (T(3,J), J=1,5)
20    FORMAT (1X,5G14.6)
```

Here's a shorter cut:

```
      PRINT 20, ((T(I,J), J=1,5), I=1,3)
20    FORMAT (1X,5G14.6)
```

To read the table, say:

```
      READ 10, ((T(I,J), J=1,5), I=1,3)
```

DATA

This example shows FORTRAN's DATA statement, which differs from BASIC's:

```
      DATA X/8.7/, Y/1.4/, Z/9.0/     X is 8.7, Y is 1.4, and Z is 9.0.
      X=100.6                         X changes to 100.6.
      PRINT 10, X,Y,Z                 The computer will print:
10    FORMAT (1X,3G14.6)                 100.600       1.40000       9.00000
      END
```

In that DATA statement, you must write 9.0, not 9: the number must be real, since Z is real.

The DATA statement resembles these three statements—

```
X=8.7
Y=1.4
Z=9.0
```

but is faster.

Here's another way to type the DATA statement:

```
DATA X,Y,Z/8.7,1.4,9.0/
```

It says X, Y, and Z are 8.7, 1.4, and 9.0 respectively.

Like the DIMENSION statement, the DATA statement belongs at the very top of the program. If you want both a DIMENSION statement and a DATA statement, put the DIMENSION statement first.

This DATA statement says A, B, C, D, and E are all 1.7, and X, Y, and Z are all 9.6:

```
DATA A,B,C,D,E,X,Y,Z/1.7,1.7,1.7,1.7,1.7,9.6,9.6,9.6/
```

Since the first 5 numbers are 1.7, and the next 3 numbers are 9.6, you can write more briefly:

```
DATA A,B,C,D,E,X,Y,Z/5*1.7,3*9.6/
```

To make A be this list—

81.7
92.6
25.3
49.8
72.1
68.8

begin your program with these lines:

```
DIMENSION A(6)
DATA A/81.7,92.6,25.3,49.8,72.1,68.8/
```

To make T be this table—

8.4 9.7
5.1 6.8
2.5 7.2
6.3 9.8

begin your program with these lines:

```
DIMENSION T(4,2)
DATA T/8.4,5.1,2.5,6.3,9.7,6.8,7.2,9.8/
```

Notice you must list the entire first column, then the second.

STRING VARIABLES

This program works on most computers:

```
      N='UP'
      PRINT 10, N
10    FORMAT (1X,A2)
      END
```

The top line says N is the string 'UP'. The next line says to print N, using the format in line 10. The A2 format means a 2-character string. (The A is derived from the word Alphabet.) The computer will print:

```
UP
```

Some computers allow apostrophes on in FORMAT, DATA, and CALL statements. (You'll learn about CALL statements later.) On such computers, the statement N='UP' is illegal. Instead say:

```
      DATA N/'UP'/
```

A string statement should look like an integer: it should begin with the letter I, J, K, L, M, or N. You can say N='UP' but shouldn't say X='UP'.

The string a variable stands for must be short. You can say N='UP' but not N='SUPERCALIFRAGILISTICEXPIALIDOCIOUS'. The longest permissible string depends on your computer:

Computer	Longest permissible string
PDP-11 (using FORTRAN IV)	2 characters
PDP-11 (using FORTRAN IV-PLUS), IBM	4 characters
PDP-10, PDP-20, Honeywell	5 characters
CDC	10 characters

Example:

```
      PRINT 10
10    FORMAT (1X,'DO YOU LIKE ME?')
      READ 20, IREPLY
20    FORMAT (A1)
      IF (IREPLY .EQ. 'Y') PRINT 30
30    FORMAT (1X,'I LIKE YOU TOO!')
      PRINT 40
40    FORMAT (1X,'SO LONG, BUSTER.')
      END
```

The first pair of lines make the computer print DO YOU LIKE ME? The next pair set IREPLY equal to the first letter the human types. If the human types YESIREE, the computer will set IREPLY equal to 'Y' and will therefore print:

```
I LIKE YOU TOO!
SO LONG, BUSTER.
```

But if the human types NOT AT ALL, the computer will set IREPLY equal to 'N' and will therefore print just:

```
SO LONG, BUSTER.
```

In that program, the string is called IREPLY instead of REPLY, to make it an integer.

Advanced example:

```
      DIMENSION NAME(25)
      PRINT 10
10    FORMAT (1X,'WHAT IS YOUR NAME?')
      READ 20, NAME
20    FORMAT (25A2)
      PRINT 30, NAME
30    FORMAT (1X,'I HATE ANYONE NAMED ',25A2)
      END
```

The top line says NAME will be a list of 25 elements. The next pair of lines print WHAT IS YOUR NAME? If the human answers—

```
BARTHOLOMEW HIERONYMOUS MCGILLICUDDY, M.D.
```

the format in line 20 sets NAME equal to 25 two-character strings:

```
NAME(1) is 'BA'     NAME(10) is 'YM'     NAME(19) is ', '
NAME(2) is 'RT'     NAME(11) is 'OU'     NAME(20) is 'M.'
NAME(3) is 'HO'     NAME(12) is 'S '     NAME(21) is 'D.'
NAME(4) is 'LO'     NAME(13) is 'MC'     NAME(22) is ' '
NAME(5) is 'ME'     NAME(14) is 'GI'     NAME(23) is ' '
NAME(6) is 'W '     NAME(15) is 'LL'     NAME(24) is ' '
NAME(7) is 'HI'     NAME(16) is 'IC'     NAME(25) is ' '
NAME(8) is 'ER'     NAME(17) is 'UD'
NAME(9) is 'ON'     NAME(18) is 'DY'
```

Format 30 prints:

```
I HATE ANYONE NAMED BARTHOLOMEW HIERONYMOUS MCGILLICUDDY, M.D.
```

To make that program run faster, use fewer strings. For example, if you have a CDC computer, each string can be as long as 10 characters, so you need only 5 strings to make 50 characters:

```
→→      DIMENSION NAME(5)
        PRINT 10
  10    FORMAT (1X,'WHAT IS YOUR NAME?')
        READ 20, NAME
→→20    FORMAT (5A10)
        PRINT 30, NAME
→→30    FORMAT (1X,'I HATE ANYONE NAMED ',5A10)
        END
```

On computers having FORTRAN 77, and on IBM computers using WATFIV, you can request super-long strings:

```
→→      CHARACTER*50 NAME
        PRINT 10
  10    FORMAT (1X,'WHAT IS YOUR NAME?')
        READ 20, NAME
→→20    FORMAT (A50)
        PRINT 30, NAME
→→30    FORMAT (1X,'I HATE ANYONE NAMED ',A50)
        END
```

The top line requests that NAME be a 50-character string. Like the DIMENSION statement, the CHARACTER statement must be put at the very top of the program, above even the DATA statements.

fancy formats

DOUBLE APOSTROPHE

To make the computer print an apostrophe, use two apostrophes next to each other.

```
      PRINT 10
10    FORMAT (1X,'MOMMY ISN''T HERE')
      END
```

The computer will print:

```
MOMMY ISN'T HERE
```

T FORMAT

You saw this program for meatball lovers:

```
      PRINT 10
10    FORMAT (1X,'EAT A',8X,'MEATBALL')
      END
```

It made the computer print EAT A, then 8 blank spaces, then MEATBALL:

```
EAT A        MEATBALL
```

This program does the same thing:

```
      PRINT 10
10    FORMAT (1X,'EAT A',T15,'MEATBALL')
      END
```

It makes the computer print EAT A, then Tab over to the 15th position on the line, and print MEATBALL. When the computer tabs to the 15th position, it considers the carriage control to be the first "position"; the E in EAT is the first character the computer will print, and the computer considers it to be the second "position"; the A in EAT is the second character the computer will print, and the computer considers it to be the third "position"; the M in MEATBALL is the 14th character the computer will print, since the T15 says it is the 15th "position".

BLANK RECORDS

Example:

```
      PRINT 10
10    FORMAT (1X,'PLEASE'/1X,'NIBBLE'//1X,'MY'/////1X,'CHEESE')
      END
```

Line 10 makes the computer print several lines. The first line will say PLEASE. The next line will say NIBBLE. The next line will be blank. The next line will say MY. The next 4 lines will be blank. The last line will say CHEESE. So altogether, the computer will print:

```
PLEASE
NIBBLE

MY




CHEESE
```

REPEATED FORMATS

The computer can feel very depressed:

```
      PRINT 10
10    FORMAT (1X,'I FEEL ',3('DOWN'))
      END
```

Line 10 is an abbreviation for this format:

```
1X,'I FEEL ','DOWN','DOWN','DOWN'
```

The computer will print:

```
I FEEL DOWNDOWNDOWN
```

Let's burp and pray:

```
      PRINT 10
10    FORMAT (1X,'JACK ',2('BURPS'/1X,'MARY '),'ALSO PRAYS')
      END
```

Line 10 is an abbreviation for this format:

```
1X,'JACK ','BURPS'/1X,'MARY ','BURPS'/1X,'MARY ','ALSO PRAYS'
```

The computer will print:

```
JACK BURPS
MARY BURPS
MARY ALSO PRAYS
```

E FORMAT

For real numbers, the usual formats are F and G, but another option is E. If you say E14.6 instead of G14.6, the computer will print an E in the answer. Here are examples:

Using G14.6 format	Using E14.6 format
▮-0.283941E-29	▮-0.283941E-29
▮▮0.293027▮▮▮▮	▮▮0.293027E+00
▮▮▮5.34523▮▮▮▮	▮▮0.534523E+01
▮▮▮39.4539▮▮▮▮	▮▮0.394539E+02
▮▮▮47802.3▮▮▮▮	▮▮0.478023E+05
▮▮▮986327.▮▮▮▮	▮▮0.986327E+06
▮▮0.288341E+24	▮▮0.288341E+24

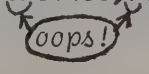
The G14.6 format is easier for a human to read than E14.6. But most programmers are stupid, don't know about G14.6, and use E14.6 instead.

P FORMAT

FORTRAN's notation differs from BASIC. If you ask the computer to print 28834100000000000000000.0 in FORTRAN, by using G14.6 (or E14.6), the computer will normally print a 0 before the decimal point, like this: 0.288341E+24. In BASIC, the computer will print a non-zero digit before the decimal point, like this: 2.88341E+23.

If you're writing a program in FORTRAN, but you prefer BASIC's notation, ask for 1PG14.6 (or 1PE14.6). The 1P makes the computer imitate BASIC. But if a FORMAT contains 1PG, it must not contain F afterwards; this will print a wrong answer:

```
FORMAT (1X,1PG14.6,F8.2)
```



Functions

SQUARE ROOT

This program finds the square root of 9:

```
      A=SQRT(9.0)
      PRINT 10, A
10    FORMAT (1X,G14.6)
      END
```

The computer will print:

```
 3.00000
```

In that program, you must say SQRT(9.0), not SQRT(9). If you say SQRT(9), the computer will either gripe or print a wrong answer.

The number in parentheses must be real. That's why you can say SQRT(9.0) but not SQRT(9). You can say SQRT(8.0+1.0) but not SQRT(8+1). You can say SQRT(X) but not SQRT(J).

Be careful when you translate from BASIC to FORTRAN: in BASIC, you say SQR; in FORTRAN, you say SQRT instead.

This program prints a table, showing the square root of 2.0, the square root of 3.0, the square root of 4.0, the square root of 5.0, etc.:

```
      X=2.0
10    Y=SQRT(X)
      PRINT 20, X,Y
20    FORMAT (1X,2G14.6)
      X=X+1.0
      GO TO 10
      END
```

The computer will print:

```
 2.00000       1.41421
 3.00000       1.73205
 4.00000       2.00000
 5.00000       2.23607
 6.00000       2.44949
 7.00000       2.64575
 8.00000       2.82843
 9.00000       3.00000
 10.0000       3.16228
 etc.
```

FLOAT

If you say FLOAT, the computer will create a FLOATing-point number (in other words, a real number), by using an integer. For example, FLOAT(3) is 3.0. If J is 7, then FLOAT(J) is 7.0.

The word FLOAT can help you reach the following goals....

GOAL: find the square root of J.
DIFFICULTY: you aren't allowed to say SQRT(J), because what you take the square root of must be a real number; you can say SQRT(X) but not SQRT(J).
SOLUTION: say X=J, and then say SQRT(X).
SHORTER SOLUTION: say SQRT(FLOAT(J)).

GOAL: divide I by J accurately.
DIFFICULTY: saying I/J doesn't solve the problem, because when the computer divides integers it gives an integer answer, instead of an accurate real answer.
SOLUTION: say X=I, then Y=J, then X/Y.
SHORTER SOLUTION: say FLOAT(I)/FLOAT(J).

RANDOM NUMBERS

Here's how to set R equal to a random decimal between 0 and 1:

Computer	What to say
CDC	R=RANF(0)
PDP-10, PDP-20	R=RAN(0)
PDP-11	R=RAN(ISEED,ISEED2)

To randomize the random numbers on PDP-10 and PDP-20 computers, put these lines near the top of your program:

```
CALL TIME(ISEED,ISEED2)
CALL SETRAN(MOD(ISEED/2+ISEED2/2, 2147483648))
```

On other computers, randomizing is even more complicated; ask the people who run your computer center.

MAXIMA AND MINIMA

To find the maximum real number in a list, ask for AMAX1. For example, AMAX1(4.7, 2.8, 41.6, 9.2, 82.3, 9.7) is 82.3. And AMIN1(4.7, 2.8, 41.6, 9.2, 82.3, 9.7) is the minimum, which is 2.8. If the numbers in the list are integers, say MAX0 instead of AMAX1, and say MIN0 instead of AMIN1.

zero, not oh

ABSOLUTE VALUE

ABS(X) means the ABSolute value of X; in other words, X without its minus sign. For example:

ABS(-5.2) is 5.2
ABS(-7.0) is 7.0
ABS(9.3) is 9.3

For integers, say IABS instead of ABS. For example, IABS(-7) is -7.

REMAINDER

When you divide 11 by 4, the remainder is 3:

```
  2
4)11
 -8
  3 is the remainder
```

If you ask for MOD(11,4) the computer will divide 11 by 4 and get the remainder, which is 3; so MOD(11,4) is 3.

Use MOD for integers; use AMOD for reals. For example, AMOD(11.0, 4.0) is 3.0.

TRIGONOMETRY

If your computer has FORTRAN 77, or your computer is an IBM having FORTRAN H-EXTENDED, or your computer is a PDP-11 having FORTRAN IV-PLUS, you can use these trigonometric functions:

Symbol	Meaning
SIN(X)	the SINe of X radians
COS(X)	the COSine of X radians
TAN(X)	the TANgent of X radians
ASIN(X)	the ArcSINe of X in radians; the number whose sine is X
ACOS(X)	the ArcCOSine of X in radians; the number whose cosine is X
ATAN(X)	the ArcTANgent of X in radians; the number whose tangent is X
SINH(X)	the SINe Hyperbolic of X
COSH(X)	the COSine Hyperbolic of X
TANH(X)	the TANgent Hyperbolic of X

If your computer is old-fashioned, it restricts you:

Old-fashioned computer	Restriction
PDP-10, PDP-20	you can't say TAN(X)
CDC	you can't say SINH(X) or COSH(X)
other IBM computers	say ARSIN(X), not ASIN(X); say ARCOS(X), not ACOS(X)
FORTRAN IV computers	you can't say TAN(X), ASIN(X), ACOS(X),SINH(X), or COSH(X)

You can replace X by any *real* number. For example, you can say SIN(4.0) but not SIN(4); you can say SIN(Y) but not SIN(J).

Be careful when you translate from BASIC to FORTRAN: in BASIC, you say ATN; in FORTRAN, you say ATAN instead.

You've seen that SIN(X) is the sine of X *radians*. But what's the sine of X *degrees*? On PDP-10 and PDP-20 computers, you can find the sine of X degrees by asking for SIND(X); and you can find the cosine of X degrees by asking for COSD(X).

ATAN2(Y,X) is about the same as ATAN(Y/X), but is faster, more accurate, and gives a useful answer even when X is zero or negative. ATAN2(Y,X) is the angle (in radians) of the line that goes through the origin and the point (X,Y).

CALCULUS

You can use these functions:

Function	Meaning
EXP(X)	e^X
ALOG(X)	$\log_e X$
ALOG10(X)	$\log_{10} X$

You can replace X by any real number, but not by an integer. Each of those functions produces a *real* answer, since none of them begins with I, J, K, L, M, or N. The logarithm function is called ALOG instead of LOG, to avoid beginning with L. Be careful when you translate from BASIC to FORTRAN: in BASIC, you say LOG; in FORTRAN, you say ALOG instead.

Exotic features

DOUBLE PRECISION

Some computers are more accurate than others:

Computer	Accuracy for real numbers
PDP-11, IBM	7 digits
PDP-10, PDP-20, Honeywell	8 digits
CDC	14 digits

For example, suppose you feed this program to a PDP-11 or IBM:

```
      A=5398.1642376236
      PRINT 10, A
10    FORMAT (1X,F15.10)
      END
```

Expect the first 7 digits the computer prints to be correct (5398.164), but the remaining digits it prints to be wrong: they arise from round-off error inside the computer.

A PDP-11 or IBM prints *some* numbers to an accuracy of 8 digits and others to an accuracy of only 6 digits, but 7 digits is typical.

For real numbers, I recommend you use a G14.6 format, because it's safe: it prints just the first 6 digits. If you want to see further digits that are probably correct, use these formats instead:

Computer	Format	What the format does
PDP-11, IBM	G15.7	prints the first 7 digits
PDP-10, PDP-20, Honeywell	G16.8	prints the first 8 digits
CDC	G22.14	prints the first 14 digits

You can obtain extra accuracy, by requesting *double precision*:

Computer	Accuracy for double precision numbers	Format
PDP-11	14 digits	D22.14
PDP-10 (using KA), Honeywell	16 digits	D24.16
IBM	17 digits	D25.17
PDP-10 (using KI or KL), PDP-20	18 digits	D26.18
CDC	29 digits	D37.29

Each of these programs computes the square root of 6.3×10^8 on a PDP-11 computer:

Using reals

```
      A=SQRT(6.3E8)
      PRINT 10, A
10    FORMAT (1X,G15.7)
      END
```

Using double precision

```
      DOUBLE PRECISION A
      A=DSQRT(6.3D8)
      PRINT 10, A
10    FORMAT (1X,D22.14)
      END
```

The program on the left computes 7 digits. The program on the right computes 14. Comparing the programs, you'll notice four differences:

1. For double precision, you must use double precision numbers. Instead of saying 6.3E8, say 6.3D8. The D means Double precision.

Real number	Double precision number
6.3E8	6.3D8
29.6	29.6D0

↖this is a zero

2. For double precision, you must use double precision functions. Instead of saying SQRT, say DSQRT.

Real function	Double precision func.
SQRT	DSQRT
AMAX1	DMAX1
ABS	DABS
AMOD	DMOD
EXP	DEXP
ALOG	DLOG
ALOG10	DLOG10

Real function	Double precision func.
SIN	DSIN
COS	DCOS
TAN	DTAN
ASIN	DASIN
ACOS	DACOS
ATAN	DATAN
ATAN2	DATAN2
SINH	DSINH
COSH	DCOSH
TANH	DTANH

3. For double precision, you must use double precision variables. Normally, the variable A would be real; to make it double precision instead, say:

```
DOUBLE PRECISION A
```

Normally, the variables LENGTH and MASS would be integers, and VOLUME would be real; to make them all double precision, say:

```
DOUBLE PRECISION LENGTH,MASS,VOLUME
```

Like the DIMENSION and CHARACTER statements, the DOUBLE PRECISION statement must be put at the very top of the program, above even the DATA statements. If you say—

```
IMPLICIT DOUBLE PRECISION(D)
```

every variable whose first letter is D will automatically be double precision; for example, DISTAN and DIAMET and DSIZE will automatically be double precision.

4. For double precision you must use double precision formats. Instead of a G format, use D22.14, or whichever D format is appropriate for your computer.

Although double precision arithmetic is more precise than real arithmetic, it's also more expensive: it consumes more of the computer's time, and the numbers consume more of the computer's memory.

If you combine an integer or a real number with a double precision number, the answer will be double precision.

COMPLEX

In mathematics, the square root of -1 is called i. So i^2 is -1. The number i obeys most of the rules of algebra:

i+i is 2i
2i+3i is 5i
(8+2i) + (7+3i) is 15+5i
(8+2i) * (7+3i) is 8*7 + 8*3i + 2i*7 + 2i*3i, which is 56 + 24i + 14i + $6i^2$, which is 56 + 24i + 14i + -6, which is 50+38i.

The number i is neither positive nor negative nor zero; it is pictured instead as being above the real number line:

```
                i

-5 -4 -3 -2 -1  0  1  2  3  4  5
```

2i is further above the real number line: it is 2 units above 0. Another example: 5+2i is 2 units above 5.

A number that involves i is called *complex*. So 5+2i is complex. Its *real part* is 5, its *imaginary part* is 2, and its *conjugate* is 5-2i.

This program multiplies 8+2i by 7+3i and prints the correct answer, 50+38i:

```
      COMPLEX B
      B=(8.0, 2.0) * (7.0, 3.0)
      PRINT 10, B
10    FORMAT (1X,2G14.6)
      END
```

FORTRAN says (8.0, 2.0) instead of 8+2i; the decimal points and parentheses are required. To make B complex instead of real, say COMPLEX B; like the DIMENSION and CHARACTER and DOUBLE PRECISION statements, the COMPLEX statement must be put at the very top of the program, above even the DATA statements. To print B, use the G14.6 format twice (once for the real part, and once for the imaginary part). The computer will print:

```
 50.0000        38.0000
```

If you say IMPLICIT COMPLEX(C), every variable whose first letter is C will automatically be complex.

Although you can write 8+2i as (8.0, 2.0), you cannot write X+Yi as (X, Y); instead write CMPLX(X, Y). The rule is: to build a complex number from variables instead of from constants, say CMPLX. The variables you use to build it must be real.

If B is complex, you can use these functions:

Function	Meaning
CSQRT(B)	the complex number that's the square root of B
CSIN(B)	the complex number that's the sine of B
CCOS(B)	the complex number that's the cosine of B
CEXP(B)	the complex number that's e^B
CLOG(B)	the complex number that's log B
CABS(B)	the *real* number that's the absolute value of B
REAL(B)	the *real* number that's the real part of B
AIMAG(B)	the *real* number that's the imaginary part of B
CONJG(B)	the complex number that's the conjugate of B

This program finds the square root of -9 and prints the correct answer, 3i:

```
      COMPLEX B,Z
      B=-9
      Z=CSQRT(B)
      PRINT 10, Z
10    FORMAT (1X,2G14.6)
      END
```

Since the top line says B is complex, the second line makes B the complex number -9+0i. The next line makes Z the square root of -9+0i, which is 0+3i. The computer will print the 0 and the 3:

```
 0.000000        3.00000
```

Since that program sets B equal to -9+0i, which FORTRAN writes as (-9.0, 0.0), you can shorten the program to this:

```
      COMPLEX Z
      Z=CSQRT( (-9.0, 0.0) )
      PRINT 10, Z
10    FORMAT (1X,2G14.6)
      END
```

Make sure the number inside CSQRT's parentheses is complex: you need the decimal points, comma, and parentheses.

When the computer combines a complex number with an integer, a real, or a double precision number, the result is complex.

Some computers limit your use of complex numbers, by restrictions such as:

"Don't say a complex number equals a double precision number",
"Don't combine a complex number with a double precision number",
"Don't raise a number to a complex power",
"Don't raise a complex number to a power, unless the power is an integer".

Find out whether your computer has those restrictions.

SUBROUTINES

This program is a combination of two *routines*:

```
      PRINT 10
10    FORMAT (1X,'KIDS')
      CALL YUMMY
      PRINT 20
20    FORMAT (1X,'LOLLIPOPS')
      END

      SUBROUTINE YUMMY
      PRINT 10
10    FORMAT (1X,'SUCK')
      RETURN
      END
```

Each routine ends with the word END. The first routine is called the *main routine*; the second routine is called the *subroutine*, and begins with the word SUBROUTINE; I decided to name it YUMMY, so its top line says SUBROUTINE YUMMY. Above the word SUBROUTINE, I put a blank line. The blank line is optional; it helps humans find where the subroutine begins.

In the main routine, the top pair of lines makes the computer print:

```
KIDS
```

The next line says CALL YUMMY, which makes the computer skip down to subroutine YUMMY. Subroutine YUMMY makes the computer print—

```
SUCK
```

and then RETURN to the main routine, which finishes by printing:

```
LOLLIPOPS
```

Altogether, the program prints:

```
KIDS
SUCK
LOLLIPOPS
```

Notice that line 10 in the main routine is different from line 10 in the subroutine. Similarly, an X in the main routine is different from one in the subroutine:

```
      X=3.4
      CALL FUNNY
      PRINT 10, X
10    FORMAT (1X,G14.6)
      END

      SUBROUTINE FUNNY
      X=925.1
      Y=X+1.0
      PRINT 10, Y
10    FORMAT (1X,G14.6)
      RETURN
      END
```

The computer will set the main routine's X equal to 3.4. Then it will call FUNNY. The X in FUNNY is 925.1, so Y is 926.1, and the computer will print:

```
  926.100
```

When the computer returns to the main routine, it will print the main routine's X, which is still:

```
  3.40000
```

To pass information from one routine to another, put the information in parentheses:

```
      A=5.2
      CALL LEMON(A)
      PRINT 10, A
10    FORMAT (1X,G14.6)
      END

      SUBROUTINE LEMON(X)
      PRINT 100, X
100   FORMAT (1X,G14.6)
      X=7.1
      RETURN
      END
```

The computer sets A equal to 5.2. Then it calls LEMON. The A and X in parentheses mean: the main routine's A is the subroutine's X, so the subroutine's X is 5.2. Line 100 prints:

```
5.20000
```

The next line changes X to 7.1. When the computer returns to the main routine, the main routine's A is the subroutine's X, so the main routine's A is 7.1. Line 10 prints:

```
7.10000
```

A harder example:

```
      P=2.1
      CALL JUNK(P)
      PRINT 10, P
10    FORMAT (1X,G14.6)
      END

      SUBROUTINE JUNK(P)
      P=3.0*P
      RETURN
      END
```

The computer sets P equal to 2.1 and then calls JUNK. The P in parentheses means: the main routine's P is the subroutine's P. The subroutine tripes P, so P becomes 6.3. When the computer returns to the main routine, line 10 prints:

```
6.30000
```

Another example:

```
      Q=1.4
      CALL FAT(5.0, Q+.3, R)
      PRINT 10, R
10    FORMAT (1X,G14.6)
      END

      SUBROUTINE FAT(X, Y, Z)
      Z=X+Y
      RETURN
      END
```

When the main routine calls FAT, the subroutine's X is 5.0; Y is Q+.3, which is 1.7; and Z is R, which is undefined. The subroutine sets Z equal to X+Y, which is 6.7. When the computer returns to the main routine, the main routine's R is the subroutine's Z, which is 6.7. Line 10 prints:

```
6.70000
```

In that CALL, you must say 5.0, not 5, since X must be real. Saying 5 will confuse the computer and make it print a wrong answer.

This subroutine finds the sum and average of three real numbers— X, Y, and Z:

```
      SUBROUTINE STAT(X, Y, Z, SUM, AVERAG)
      SUM=X+Y+Z
      AVERAG=SUM/3.0
      RETURN
      END
```

This main routine uses STAT to find the sum and average of 8.1, 2.6, and 9.3:

```
      CALL STAT(8.1, 2.6, 9.3, SUM, AVERAG)
      PRINT 10, SUM,AVERAG
10    FORMAT (1X,2G14.6)
      END
```

This subroutine finds the sum and average of three *double precision* numbers:

```
      SUBROUTINE DSTAT(X, Y, Z, SUM, AVERAG)
      DOUBLE PRECISION X, Y, Z, SUM, AVERAG
      SUM=X+Y+Z
      AVERAG=SUM/3D0
      RETURN
      END
```

This main routine uses DSTAT to find the sum and average of π, e, and my phone number (2668128):

```
      DOUBLE PRECISION SUM, AVERAG
      CALL DSTAT(3.1415926535898D0, 2.7182818284590D0, 2668128D0, SUM,
     6AVERAG)
      PRINT 10, SUM, AVERAG
10    FORMAT (1X,2D22.14)
      END
```

You must say DOUBLE PRECISION in both the main routine and the subroutine.

This subroutine finds the sum and average of X_1, X_2, X_3, ..., X_N:

```
      SUBROUTINE STAT2(X, N, SUM, AVERAG)
      DIMENSION X(N)
      SUM=0.0
      DO 10 I=1,N
         SUM=SUM+X(I)
10    CONTINUE
      AVERAG=SUM/N
      RETURN
      END
```

This main routine uses STAT2 to find the sum and average of 8.4, 9.6, 20.1, 7.2, 91.5, and 3.6:

```
      DIMENSION X(6)
      DATA X/8.4,9.6,20.1,7.2,91.5,3.6/
      CALL STAT2(X, 6, SUM, AVERAG)
      PRINT 10, SUM,AVERAG
10    FORMAT (1X,2G14.6)
      END
```

You must put the DIMENSION statement in both the main routine and the subroutine. In the main routine, the DIMENSION must be a constant (6); in the subroutine, the DIMENSION can be a variable (N).

This main routine asks you to input some numbers, and then prints their sum and average, by using STAT2:

```
      DIMENSION X(100)
      PRINT 10
10    FORMAT (1X,'HOW MANY NUMBERS WOULD YOU LIKE TO GIVE ME?')
      READ *, N
      PRINT 20
20    FORMAT (1X,'TYPE THE NUMBERS')
      READ *, (X(I), I=1,N)
      CALL STAT(X, N, SUM, AVERAG)
      PRINT 10, SUM,AVERAG
10    FORMAT (1X,'THE SUM IS',G14.6,' AND THE AVERAGE IS',G14.6)
      END
```

That DIMENSION statement lets you input up to 100 numbers.

If a main routine says DIMENSION X(100), and X is passed between the main routine and the subroutine, the subroutine's DIMENSION statement must say no more than X(100): if the subroutine's DIMENSION statement says X(N), the N must be no more than 100.

For *double* subscripts, the main routine's DIMENSION statement must be *exactly* the same as the subroutine's. For example, if the main routine says DIMENSION X(25,4), the subroutine must say DIMENSION X(25,4) also: it must not say DIMENSION X(20,3); if it says DIMENSION X(M,N), the M must be exactly 25.

The *Scientific Subroutine Package (SSP)* is a collection of subroutines written by IBM that do statistics, calculus, equation-solving, and advanced math. The *Calcomp subroutines* make the computer operate a Calcomp plotter (a device that draws fancy shapes on paper, by using a felt-tip or ballpoint pen). Most large computers store the SSP and Calcomp subroutines on disk permanently. Ask the people in your computer center how to combine your own main routines with those subroutines, and how to put your *own* library of subroutines onto the disk.

If you and your friends want to write a big program together, divide the problem into a main routine and several subroutines, and have each person write one routine. Although you'll need a group conference to decide what the SUBROUTINE and CALL statements will be, you don't have to agree on the names of variables, since the names of subroutine variables have nothing to do with the names of main-routine variables.

YOUR OWN FUNCTIONS

Instead of using functions such as SQRT and ABS, you can invent your *own* functions. Here's how to invent a function called F, and how to use the function you've invented:

```
      A=F(5.2)+1.0
      PRINT 10, A
10    FORMAT (1X,G14.6)
      END

      FUNCTION F(X)
      F=2.0*X
      RETURN
      END
```

The program consists of two routines: the first routine is the *main* routine; the second routine is the definition of the FUNCTION F. Each routine ends with the word END. The main routine's top line requires the computer to find F(5.2), so the computer hunts for the definition of FUNCTION F. The definition say F is twice 5.2, which is 10.4. So A is 11.4. The computer will print:

```
 11.40000
```

Like a subroutine, a function definition can be very long and have many variables in parentheses. The function's name can be F or G or any other name, such as MASS. If the function's name begins with I, J, K, L, M, or N, the computer will assume its value is an integer. If you want the function MASS(X) to be double precision instead of an integer, begin your main routine by saying—

```
      DOUBLE PRECISION MASS
```

and begin your function definition with this line:

```
      DOUBLE PRECISION FUNCTION MASS(X)
```

FILES

The computer can write its answers onto a data file:

```
      A=14.6+75.2
      WRITE(3,10) A
10    FORMAT (1X,G14.6)
      WRITE(7,20)
20    FORMAT (1X,'TINA, PLEASE TICKLE MY TUBA')
      END
```

Since A is 89.8, the second line makes the computer write 89.8 onto "file 3", using line 10's format. The next pair of lines make the computer write onto "file 7", using line 20's format; so the computer will write TINA, PLEASE TICKLE MY TUBA onto file 7.

Before running that program, tell the computer where to put files 3 and 7. You can make the computer put them on a disk, line printer, tape, terminal, or wherever else you please. Ask the people in your computer center how to tell the computer where to put the files.

If your program has many statements saying to write onto file 3, the computer will write many lines onto file 3, so the file will become long.

Use carriage controls only if the file is on a line printer or terminal.

If you say—

```
READ(4,30) X
```

the computer will read the value of X from file 4, using line 30's format. If you say file 4 is the card reader, the computer will wait for you to feed in a card; if you say file 4 is your terminal, the computer will wait for you to type on the terminal; if you say file 4 is on a tape or disk, the computer will assme you created the file before running the program.

For handling files, the word READ works even on PDP computers.

For PDP-10 and PDP-20 computers, here's how to make file 3 be on disk and named JOE...Near the beginning of your program, say:

```
OPEN(UNIT=3, DEVICE='DSK', FILE='JOE.')
```

Near the end of your program, say:

```
CLOSE(UNIT=3)
```

PASCAL

Imagine that you're running for President, against Calvin Coolidge. To win, you must pass Cal in the polls.

You must pass Cal. "Pass Cal" is the correct way to pronounce "PASCAL", the name of the computer language that programmers are falling in love with.

Of the four popular languages— FORTRAN, COBOL, BASIC, and PASCAL— the newest, most modern, and most elegant is PASCAL. Many programmers who've used FORTRAN are switching to PASCAL. On microcomputers, the two most popular languages are BASIC and PASCAL. Many professors feel PASCAL does a better job of training a student's mind than BASIC, FORTRAN, or COBOL.

Unfortunately, PASCAL is harder to learn than BASIC. But if you learn PASCAL and write your programs in PASCAL, your programs will become more neatly organized, and will therefore contain fewer hidden errors than if written in BASIC.

The original version of PASCAL was invented by Niklaus Wirth in Switzerland, for a CDC maxicomputer. Later, versions of PASCAL were developed for minicomputers and microcomputers. The most popular microcomputer version was invented at the University of California at San Diego (UCSD), and is called *UCSD PASCAL*. Another version is called *ANSI standard PASCAL*. In this chapter, I'll steer a middle course between those versions.

Fun

SIMPLE PROGRAMS

Here's a simple PASCAL program, called WINE:

```
PROGRAM WINE;
BEGIN;
WRITELN('I WOULD LIKE TO KISS');
WRITELN('YOUR BOTTLE OF WINE');
END.
```

The top line says the program is called WINE. The guts of the program begins with the word BEGIN, and ends with the word END. Every line ends with a semicolon, except that the bottom line ends with a period.

When you run the program, the computer will write this on your terminal:

```
I WOULD LIKE TO KISS
YOUR BOTTLE OF WINE
```

Weird computers

On CDC computers, the top line should say INPUT and OUTPUT, like this:

```
PROGRAM WINE(INPUT,OUTPUT);
```

On PDP-10 and PDP-20 computers, each WRITELN statement should say TTY, like this:

```
WRITELN(TTY,'I WOULD LIKE TO KISS');
WRITELN(TTY,'YOUR BOTTLE OF WINE');
```

MULTIPLE WRITING

This program belongs in your bathroom:

```
PROGRAM SMELLY;
BEGIN;
WRITELN('FAR','TIN','G');
END.
```

The computer will write FAR, TIN, and G all on the same line, like this:

```
FARTING
```

On PDP-10 and PDP-20 computers, the WRITELN must say TTY, like this:

```
WRITELN(TTY,'FAR','TIN','G');
```

MATH

Like FORTRAN, PASCAL uses integers and reals.

An integer has no decimal point. Here are some integers:

```
723          0          -4
```

A real has a decimal point; for example, -28.3 is a real. You must put digits before the decimal point, and digits after it:

Correct reals	Incorrect
0.37	.37
5.0	5.
6.0E24	6E24

In that table, the entries marked "incorrect" are incorrect in PASCAL, even though they're okay in FORTRAN.

Like FORTRAN, PASCAL uses these four operators:

```
+     -     *     /
```

If you combine a real with a real, the answer is real. If you combine a real with an integer, the answer is still real. If you combine an integer with an integer, the answer is an integer, *except for division*:

7+4 is the integer 11
7-4 is the integer 3
7*4 is the integer 28
7/4 is the real 1.75 (In FORTRAN, 7/4 would be the integer 1 instead.)

This program makes the computer do all that arithmetic:

```
PROGRAM MATH;
BEGIN;
WRITELN(7+4);
WRITELN(7-4);
WRITELN(7*4);
WRITELN(7/4);
END.
```

Here's what the computer will write; the symbol ▮ represents a blank space:

UCSD PASCAL

```
11
3
28
▮1.75
```

UCSD PASCAL puts a blank space before every *real* number that's not negative.

CDC PASCAL

```
▮▮▮▮▮▮▮▮11
▮▮▮▮▮▮▮▮▮▮3
▮▮▮▮▮▮▮▮28
▮+1.7500000000000E+000
```

To write an integer, CDC PASCAL writes enough blanks to make 10 characters altogether. To write a real, CDC PASCAL uses E notation: it writes a blank space before the number, 13 digits after the decimal point, and a 3-digit exponent.

Let's try writing all those numbers on the same line:

```
PROGRAM LINE;
BEGIN;
WRITELN(7+4,7-4,7*4,7/4);
END.
```

Here's what the computer will write:

UCSD PASCAL

```
11328▮1.75
```

CDC PASCAL

```
▮▮▮▮▮▮▮▮11▮▮▮▮▮▮▮▮▮▮3▮▮▮▮▮▮▮▮28▮+1.750000000000000E+000
```

PASCAL lets you use these functions:

Function	Meaning
ABS(-6)	the absolute value of -6; it's 6
SQR(3)	the square of 3; it's 9
SQRT(9)	the square root of 9; it's 3
SIN(2)	the sine of 2 radians
COS(2)	the cosine of 2 radians
ARCTAN(2)	the arctangent of 2, in radians
EXP(5)	e^5, where e is 2.71828182845904523536
LN(9)	the natural logarithm of 9; it's $\log_e 9$

The number in parentheses can be either an integer or a real. The computer's answer is usually a real (exception: for ABS and SQR, the computer's answer is an integer if the number in parentheses is an integer).

For example, this program computes the square root of 9:

```
PROGRAM ROOT;
BEGIN;
WRITELN(SQRT(9));
END.
```

The computer will write:

UCSD PASCAL

```
▮3.0
```

CDC PASCAL

```
▮+3.0000000000000E+000
```

Don't confuse SQR with SQRT. The SQR means *square*; the SQRT means square *root*. To find the square of 3, say SQR(3), or say 3*3. In BASIC, you could say 3↑2; in FORTRAN, you could say 3**2; but in PASCAL, you can't use the symbols ↑ or **; use SQR instead.

To turn a real number into an integer, use these functions:

Function	Meaning
ROUND(3.9)	3.9 rounded to the nearest integer; it's 4
TRUNC(3.9)	3.9 truncated (by removing the decimal point); it's 3

VARIABLES

You can use variables:

```
PROGRAM VARY;                          The program's name is VARIABLES.
VAR                                    Here are the variables....
        FRED,MARTHA: INTEGER;                  FRED and MARTHA are integers;
        JILL,TOM: REAL;                        JILL and TOM are reals.
BEGIN;                                 Now let's begin the program....
FRED:=2;                               FRED (which is an integer) is 2.
MARTHA:=4+5;                           MARTHA (which is an integer) is 9.
JILL:=9.2;                             JILL (which is a real) is 9.2.
TOM:=1.4+2.3;                          TOM (which is a real) is 3.7.
WRITELN(FRED*MARTHA,JILL+TOM); The computer will write 18 and 12.9.
END.
```

The second line says VAR. The word VAR doesn't have a semicolon after it; the lines underneath VAR are indented. The indentation is optional, but is a good habit. To indent, hit the space bar several times, or type a controlled I. In those indented lines, you must list all the program's variables. You must say which variables are integers and which are reals.

In PASCAL, a variable's name can be as long as you like: the name can be FRED or WASHINGTON or even SUPERCALIFRAGILISTICEXPIALIDOCIOUS. The name can consist of letters and digits, but must begin with a letter. The computer looks at just the name's first 8 characters; so if you say—

```
SUPERCALIFRAGILISTICEXPIALIDOCIOUS:=25000;
```

the computer will think you said:

```
SUPERCAL:=25000;
```

In PASCAL, you must put a colon in front of the equal sign:

Correct	Incorrect
FRED:=2;	FRED=2;

You can't set an integer variable equal to a real. For example, if you say—

```
VAR
        ANN: REAL;
        BILL: INTEGER;
```

you can*not* say:

```
ANN:=3.9;
BILL:=ANN;
```

Instead, you must say—

```
BILL:=ROUND(ANN);
```

or

```
BILL:=TRUNC(ANN);
```

READ

This program works in UCSD PASCAL:

```
PROGRAM MYSQRT;
VAR
        X: REAL;
BEGIN;
WRITELN('WHAT IS YOUR FAVORITE NUMBER?');
READ(X);
WRITELN('ITS SQUARE ROOT IS',SQRT(X));
END.
```

When you run the program, the computer asks:

```
WHAT IS YOUR FAVORITE NUMBER?
```

Then the READ(X) statement makes the computer wait for you to input a value of X. The computer expects you to input a real number; if you input an integer instead, the computer will automatically turn it into a real. For example, if you input 9, the computer will automatically turn it into 9.0, and will write:

```
ITS SQUARE ROOT IS 3.0
```

Weird computers

CDC, PDP-10, and PDP-20 computers ignore the first line you input. So you must input a blank line, and *afterwards* input the number you want to take the square root of.

On PDP-10 and PDP-20 computers, if you say WRITELN and then READ, you must insert the word BREAK between them:

```
WRITELN('WHAT IS YOUR FAVORITE NUMBER?');
BREAK;
READ(X);
```

IF

The computer can criticize your age:

```
PROGRAM KIDDO;
VAR
        AGE: INTEGER;
BEGIN;
WRITELN('HOW OLD ARE YOU?');
READ(AGE);
IF AGE<18 THEN
        BEGIN;
        WRITELN('YOU ARE STILL A MINOR.');
        WRITELN('AH, THE JOYS OF YOUTH!');
        END;
WRITELN('GLAD TO MEET YOU.');
END.
```

The computer asks:

```
HOW OLD ARE YOU?
```

If you input a number less than 18, the computer replies:

```
YOU ARE STILL A MINOR.
AH, THE JOYS OF YOUTH!
GLAD TO MEET YOU.
```

If you input a number that's at least 18, the computer says just this:

```
GLAD TO MEET YOU.
```

In that program the line that says "IF" is a heading: the lines underneath it are indented. Because the "IF" line is a heading, do *not* put a semicolon at the end of that line!

To make the program fancier, insert extra lines:

```
  PROGRAM MINOR;
  VAR
          AGE: INTEGER;
  BEGIN;
  WRITELN('HOW OLD ARE YOU?');
  READ(AGE);
  IF AGE<18 THEN
          BEGIN;
          WRITELN('YOU ARE STILL A MINOR.');
          WRITELN('AH, THE JOYS OF YOUTH!');
→→        END
→→ELSE
→→        BEGIN;
→→        WRITELN('I AM GLAD TO HEAR YOU ARE AN ADULT!);
→→        WRITELN('NOW WE CAN HAVE SOME ADULT FUN!');
→→        END;
  WRITELN('GLAD TO MEET YOU.');
  END.
```

If your age is less than 18, the computer will say:

```
YOU ARE STILL A MINOR.
AH, THE JOYS OF YOUTH!
GLAD TO MEET YOU.
```

If your age is *not* less than 18, the computer will say:

```
I AM GLAD TO HEAR YOU ARE AN ADULT.
NOW WE CAN HAVE SOME ADULT FUN!
GLAD TO MEET YOU.
```

In that program, the line that says "ELSE" is a heading, so do *not* put a semicolon after the "ELSE". Immediately above the word ELSE, you'll see the word END; do *not* put a semicolon after that END.

Like BASIC, PASCAL uses these symbols in the IF line:

```
<          >          =          <=          >=          <>          AND          OR
```

In the IF line, the symbol for "equals" is "="; outside the IF line, the symbol for "equals" is ":=".

Instead of saying—

```
IF I=3 OR I=8 OR I=25 OR I=95 THEN
```

you can say:

```
IF I IN [3,8,25,95] THEN
```

Loops

FOR

This program makes the computer get drunk:

```
PROGRAM BEER;
VAR
        I: INTEGER;
BEGIN;
FOR I := 6 TO 10 DO
        BEGIN;
        WRITELN('I DRANK ',I,' BOTTLES OF BEER');
        WRITELN('HOORAY!');
        END;
WRITELN('NOW I AM DEAD DRUNK');
END.
```

The computer will write:

```
I DRANK 6 BOTTLES OF BEER
HOORAY!
I DRANK 7 BOTTLES OF BEER
HOORAY!
I DRANK 8 BOTTLES OF BEER
HOORAY!
I DRANK 9 BOTTLES OF BEER
HOORAY!
I DRANK 10 BOTTLES OF BEER
HOORAY!
NOW I AM DEAD DRUNK
```

In that program, the line that says "FOR" is a heading, so do *not* put a semicolon at the end of that line.

If you want the computer to count backwards— from 10 down to 6— change the FOR line to this:

```
FOR I := 10 DOWNTO 6 DO
```

REPEAT...UNTIL

This program plays a guessing game:

```
PROGRAM GAME;
VAR
        GUESS: INTEGER;
BEGIN;
WRITELN('I AM THINKING OF A NUMBER FROM 1 TO 10');
REPEAT
        WRITELN('YOU HAVE NOT GUESSED MY NUMBER YET');
        WRITELN('WHAT IS MY NUMBER?');
        READ(GUESS);
UNTIL GUESS=6;
WRITELN('CONGRATULATIONS! YOU GUESSED IT! MY NUMBER IS 6');
END.
```

The computer begins the game by saying:

```
I AM THINKING OF A NUMBER FROM 1 TO 10
```

Then the computer repeats the following procedure: it says YOU HAVE NOT GUESSED MY NUMBER YET, asks WHAT IS MY NUMBER, waits for the human to input a guess, and checks whether the human guessed the number 6. It repeats the procedure again and again, until the human finally guesses the number 6. Then the computer says:

```
CONGRATULATIONS! YOU GUESSED IT! MY NUMBER IS 6
```

In that program, the computer repeats the indented lines, until GUESS is 6.

If a program contains the word REPEAT, it must also contain the word UNTIL. The lines between REPEAT and UNTIL are done repeatedly. After each time, the computer checks whether to repeat again.

Since the REPEAT line is a heading, it has no semicolon.

The UNTIL statement resembles an IF statement; it can use these symbols:

```
<       >       =       <=      >=      <>      AND     OR      IN
```

Here's another example. Let's make the computer start with the number 3, and keep doubling it, like this:

```
3
6
12
24
48
96
```
etc.

Let's make the computer keep doubling, but not go over 1000. So altogether, let's make the computer write:

```
3
6
12
24
48
96
192
384
768
THOSE ARE ALL THE NUMBERS BELOW 1000
```

Here's the program:

```
PROGRAM DOUBLE;
VAR
        I: INTEGER;
BEGIN;
I:=3;                   Start I at 3.
REPEAT                  Do the indented lines repeatedly, until I>=1000.
        WRITELN(I);
        I:=I*2;
UNTIL I>=1000;
WRITELN('THOSE ARE ALL THE NUMBERS LESS THAN 1000');
END.
```

WHILE

The word WHILE resembles the word REPEAT. For example, the previous program (which doubled 3 repeatedly) can be rewritten, by using WHILE:

```
  PROGRAM DOUBLE;
  VAR
          I: INTEGER;
  BEGIN;
  I:=3;                  Start I at 3.
→→WHILE I<1000 DO        If I<1000, do the indented lines; and repeat the lines
→→        BEGIN;         again and again, as long as I remains less than 1000.
          WRITELN(I);
          I:=I*2;
→→        END;
  WRITELN('THOSE ARE ALL THE NUMBERS LESS THAN 1000');
  END.
```

The indented lines say to WRITELN(I) and then to double I. The computer will repeat those indented lines many times. Before each repetition, the computer checks to make sure I is still less than 1000. When I passes 1000, the computer stops loopin, and instead proceeds to the next idea, which says to write THOSE ARE ALL THE NUMBERS LESS THAN 1000.

Notice that these two structures serve almost the same purpose:

Using REPEAT...UNTIL	Using WHILE
REPEAT	WHILE I<1000 DO
etc.	BEGIN;
UNTIL I>=1000;	*etc.*
	END;

Each of those structures makes the computer repeat the indented lines, and makes the computer check whether I is still less than 1000. If you say REPEAT...UNTIL, the check is done at the bottom of the loop, so the computer goes through the loop once before checking. If you say WHILE, the check is done at the top of the loop, so the computer checks whether I<1000 before doint the loop the first time.

Exotic features

ALTERNATIVES

Every statement must end with a semicolon. You can put several statements on the same line:

Normal	Alternative

`A:=3;`
`B:=7.6;`
`WRITELN(A+B);`
	`A:=3; B:=7.6; WRITELN(A+B);`

You can continue a statement onto the next line:

Normal	Alternative

`A:=5-2+1;`	`A:=`
	`5-2`
	`+1;`

But do *not* divide a statement in the middle of a word, number, symbol, or string:

Normal	Okay	Wrong, because middle of word	Wrong, because middle of number	Wrong, because middle of symbol
`IF AGE>=18 THEN`	`IF AGE` `>=18 THEN`	`IF AG` `E>=18 THEN`	`IF AGE>=1` `8 THEN`	`IF AGE>` `=18 THEN`

You can omit the semicolon after BEGIN:

Normal	Alternative
`BEGIN;`	`BEGIN`

You can omit a semicolon, if the next word is END:

Normal	Alternative
`A:=3;` `END;`	`A:=3` `END;`

You can omit a semicolon, if the next word is UNTIL:

Normal	Alternative
`I=I*2;` `UNTIL I>=1000;`	`I=I*2` `UNTIL I>=1000;`

Here's a short cut that's more useful: underneath a heading, if you see *only one* indented line (besides BEGIN and END), move that indented line, and put it next to the heading (and omit the BEGIN and END).

Long way

```
FOR I := 1 TO 10 DO
        BEGIN;
        WRITELN(I);
        END;

WHILE A<100 DO
        BEGIN;
        A:=A+7;
        END;

IF AGE>=65 THEN
        BEGIN;
        WRITELN('MUST RETIRE');
        END;

IF WEIGHT>=200 THEN
        BEGIN;
        WRITELN('FAT');
        END
ELSE
        BEGIN;
        WRITELN('OKAY');
        END;

REPEAT
        A:=A+7;
UNTIL A>100;
```

Short cut

```
FOR I := 1 TO 10 DO WRITELN(I);

WHILE A<100 DO A:=A+7;

IF AGE>=65 THEN WRITELN('MUST RETIRE');

IF WEIGHT>=200 THEN WRITELN('FAT') ELSE WRITELN('OKAY');

REPEAT A:=A+7 UNTIL A>100;
```

COMMENTS

Here's how to put a comment into your program:

BASIC
```
10 REM  I HATE COMPUTERS
```

FORTRAN
```
C   I HATE COMPUTERS
```

PASCAL
```
{I HATE COMPUTERS}
```

If your terminal's keyboard doesn't have braces, use parentheses and asterisks instead:

PASCAL, if no braces
```
(*I HATE COMPUTERS*)
```

Do *not* put a semicolon after the comment.

GOTO

You can say GOTO.

```
PROGRAM SKIPPER;
BEGIN;
WRITELN('MY DOG');
GOTO 10;
WRITELN('NEVER');
10: WRITELN('DRINKS WHISKEY');
END.
```

The computer will write MY DOG, then skip to line 10. Altogether, the computer will write:

```
MY DOG
DRINKS WHISKEY
```

Do *not* put a space between GO and TO.
Here's another example:

```
PROGRAM LOVEHATE;
BEGIN;
10: WRITELN('LOVE');
WRITELN('HATE');
GOTO 10;
END.
```

The computer will write LOVE and HATE repeatedly:

```
LOVE
HATE
LOVE
HATE
LOVE
HATE
etc.
```

The largest permissible line number is 9999.
In PASCAL, you'll rarely need to say GOTO; instead, try using the words IF, ELSE, FOR, REPEAT, and WHILE.

Weird computers

In UCSD PASCAL, if you want to say GO TO, you must say (*$G+*) at the top of your program:

```
(*$G+*)
PROGRAM SKIPPER;
etc.
```

PROCEDURE

This program teases you, with the help of some insults:

```
PROGRAM TEASE;

PROCEDURE INSULT;
BEGIN;
WRITELN('YOU ARE STUPID');
WRITELN('YOU ARE UGLY');
END;

(*MAIN ROUTINE*)
BEGIN;
WRITELN('WE ALL KNOW...');
INSULT;
WRITELN('...AND YET WE LOVE YOU');
END.
```

The program is called TEASE. It defines INSULT to be this procedure:

```
WRITELN('YOU ARE STUPID');
WRITELN('YOU ARE UGLY');
```

The main routine makes the computer write 'WE ALL KNOW...', then do the INSULT procedure, then write '...AND YET WE LOVE YOU'; altogether, the computer will write:

```
WE ALL KNOW...
YOU ARE STUPID
YOU ARE UGLY
...AND YET WE LOVE YOU
```

The PROCEDURE, called INSULT, is a subroutine. In PASCAL, subroutines come *before* the main routine. (In BASIC and FORTRAN, subroutines come *after* the main routine, instead.)

The PROCEDURE comes below the program's VAR:

```
PROGRAM LARGER;          The program is called LARGER.
VAR                      The program uses a variable,
      I: INTEGER;        called I, which is an integer.

PROCEDURE DOUBLE;        The subroutine is called DOUBLE.
BEGIN;
I:=2*I;                  The subroutine doubles the value of I,
WRITELN(I);              and writes the new value of I.
END;

(*MAIN ROUTINE*)         Here's the main routine....
BEGIN;
I:=7,                    I starts at 7.
DOUBLE;                  The computer does subroutine DOUBLE, which writes 14.
DOUBLE;                  The computer does subroutine DOUBLE again, which writes 28.
WRITELN(I+1);            The computer will write 29.
END.
```

CHAR

A variable can stand for a character:

```
PROGRAM RISE;
VAR
        ANN,JOAN: CHAR;      ANN and JOAN will each be a character.
BEGIN;
ANN:='U';                    ANN is the character 'U'.
JOAN:='P';                   JOAN is the character 'P'.
WRITELN(ANN,JOAN);           The computer will write 'U' and 'P'.
END.
```

The computer will write U and P like this:

```
UP
```

You can put a character variable after the word FOR:

```
PROGRAM LOOP;
VAR
        I: CHAR;
BEGIN;
FOR I := 'A' TO 'E' DO WRITELN(I);
END.
```

The computer will write:

```
A
B
C
D
E
```

SUBSCRIPTS

Suppose you want X to be this list of real numbers: 4.2, 71.6, 8.3, 92.6, 403.7, 1.4. Here's how:

```
PROGRAM LIST;
VAR
        X: ARRAY [1..6] OF REAL;
BEGIN;
X[1]:=4.2;
X[2]:=71.6;
X[3]:=8.3;
X[4]:=92.6;
X[5]:=403.7;
X[6]:=1.4;
WRITELN(X[1]+X[2]+X[3]+X[4]+X[5]+X[6]);
END.
```

The computer will write the sum, 581.8.

Subscripts can be negative:

```
PROGRAM MINUS;
VAR
        Y: ARRAY [-2..3] OF REAL;
BEGIN;
Y[-2]:=400.1;
Y[-1]:=274.1;
Y[0]:=9.2;
Y[1]:=8.04;
Y[2]:=0.6;
Y[3]:=-5.0;
WRITELN(Y[-2]+Y[-1]+Y[0]+Y[1]+Y[2]+Y[3]);
END.
```

The computer will write the sum, 687.04.

If you want Z to be a table that has 6 rows and 4 columns of reals, the VAR statement should look like this:

```
VAR
        Z: ARRAY [1..6, 1..4] OF REAL;
```

The number in the 3rd row and 2nd column is called Z[3,2]. The entire first row of Z is called Z[1]; the second row is called Z[2]; etc. For example, if you say—

```
Z[5]:=Z[3];
```

the computer will at the real numbers in the 3rd row of Z, and copy them into the 5th row. Suppose W is another array that has 6 rows and 4 columns; if you say—

```
W:=Z;
```

the computer will look at each real number in Z and copy it into W.

You can have many kinds of arrays: you can have an array of REAL, an array of INTEGER, or even an array of CHAR.

WRITING STRINGS

In UCSD PASCAL, a variable can stand for a string:

```
PROGRAM LOVER;
VAR
       X: STRING;
BEGIN;
X:='I LOVE MY MOTH';
WRITELN(X,'ER');
END.
```

The computer will write 'I LOVE MY MOTH' and then 'ER', like this:

```
I LOVE MY MOTHER
```

Weird computers

PDP computers don't understand STRING; instead of STRING, say PACKED ARRAY:

```
  PROGRAM LOVER;
  VAR
→→       X: PACKED ARRAY [1..14] OF CHAR;   That means: X is a string of 14 characters
  BEGIN;
  X:='I LOVE MY MOTH';
  WRITELN(X,'ER');
  END.
```

(On PDP-10 and PDP-20 computers, omit the word PACKED, and put TTY into the WRITELN statement.)

CDC computers don't understand STRING, and don't know how to write an array; instead of writing the array, you must write each character of the array separately, by using a FOR loop:

```
  PROGRAM LOVER(INPUT,OUTPUT);
  VAR
         X: PACKED ARRAY [1..14] OF CHAR;
→→       I: INTEGER;
  BEGIN;
  X:='I LOVE MY MOTH';
→→FOR I := 1 TO 14 DO WRITE(X[I]);
→→WRITELN('ER');
  END.
```

READING STRINGS

UCSD PASCAL lets the computer read a string:

```
PROGRAM GREET;
VAR
        X: STRING;
BEGIN;
WRITELN('WHAT IS YOUR NAME?');
READ(X);
WRITELN('HELLO ',X,' THE BEAUTIFUL');
END.
```

When you run that program, the computer asks:

```
WHAT IS YOUR NAME?
```

Then the READ(X) statement makes the computer wait for you to input a string X. If you input the name MARILYN MONROE, the computer will write:

```
HELLO MARILYN MONROE THE BEAUTIFUL
```

Weird computers

PDP and CDC computers don't understand STRING; for those computers, you must rewrite the program:

UCSD PASCAL

```
X: STRING;

READ(X);

WRITELN('HELLO ',X,
' THE BEAUTIFUL');
```

PDP and CDC, if you know X will have 14 characters

```
X: PACKED ARRAY [1..14] OF CHAR;
I: INTEGER;

FOR I := 1 TO 14 DO READ(X[I]);
READLN;

WRITE('HELLO ');
FOR I := 1 TO 14 DO WRITE(X[I]);
WRITELN(' THE BEAUTIFUL');
```

PDP and CDC, if you don't know the length of X

```
X: PACKED ARRAY [1..80] OF CHAR;
I,XLENGTH: INTEGER;

I:=0;
WHILE NOT EOLN DO
        BEGIN;
        I:=I+1;
        READ(X[I]);
        END;
READLN;
XLENGTH:=I;

WRITE('HELLO ');
FOR I := 1 TO XLENGTH DO
        WRITE(X[I]);
WRITELN(' THE BEAUTIFUL');
```

On PDP-10 and PDP-20 computers, omit the word PACKED, and say (TTY) for READ and WRITE and READLN and WRITELN and EOLN.

Like BASIC, FORTRAN, and PASCAL, COBOL is a computer language. "COBOL" is pronounced "koe ball" and stands for "COmmon Business Oriented Language".

COBOL solves *business* problems that involve *large* files of data. Therefore, COBOL is most popular among *business*men who have *maxi*computers. But today, you can use COBOL even on minicomputers and microcomputers.

In the "help wanted" section of your local newspaper, most ads that say "programmer wanted" are placed by businessmen that have maxicomputers. Therefore, most of the "programmer wanted" ads demand a knowledge of COBOL. So if you want to get a job through the "help wanted" section, a knowledge of COBOL will help you more than BASIC, FORTRAN, or PASCAL.

The first version of COBOL was called COBOL 60, because it was invented in 1960. Then came COBOL 61, COBOL 65, COBOL 68, and COBOL 74. Today, most minicomputers and microcomputers use a slight variation of COBOL 74; most maxicomputers use a compromise between COBOL 68 and COBOL 74. In this chapter, you'll learn how to write COBOL programs on today's popular computers.

During the 1960's and early 1970's, COBOL programmers used a style called "easy programming". Today, most COBOL programmers use a more sophisticated style, called "*structured* programming". This chapter explains structured programming. Though it's harder to learn than easy programming, it will make your boss kiss you.

SIMPLE PROGRAMS

Every COBOL program is written as an outline. Here's a short outline; to turn it into a COBOL program, just fill in the blanks:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
        The program's name.
AUTHOR.
        Your name.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
        The computer's name.
OBJECT-COMPUTER.
        The computer's name again.

DATA DIVISION.

PROCEDURE DIVISION.
MAIN-ROUTINE.
        What you want the computer to do.
        STOP RUN.
```

For example, here's a COBOL program I wrote:

Program	Reason
IDENTIFICATION DIVISION.	
PROGRAM-ID.	
HARRY.	The program's name is HARRY.
AUTHOR.	
RUSS WALTER.	My name is Russ Walter.
ENVIRONMENT DIVISION.	
CONFIGURATION SECTION.	
SOURCE-COMPUTER.	
DECSYSTEM-20.	My computer is a DECsystem-20 (which is a PDP-20).
OBJECT-COMPUTER.	
DECSYSTEM-20.	
DATA DIVISION.	
PROCEDURE DIVISION.	
MAIN-ROUTINE.	
DISPLAY "LIFE STINKS".	I want the computer to print "LIFE STINKS".
STOP RUN.	

When I run that program, the computer will print:

```
LIFE STINKS
```

Every COBOL program consists of four parts. The first part of the program is called the IDENTIFICATION DIVISION: it includes the program's name and the programmer's name. The second part of the program is called the ENVIRONMENT DIVISION: it includes the computer's name. The third part of the program is called the DATA DIVISION; for a simple program, the DATA DIVISION is blank. The fourth part of the program is the PROCEDURE DIVISION: it says what you want the computer to do.

The order is important: the IDENTIFICATION DIVISION must come *first*, then the ENVIRONMENT DIVISION, then the DATA DIVISION, and finally the PROCEDURE DIVISION. So to become an expert COBOL programmer, you must memorize: "IDENTIFICATION, ENVIRONMENT, DATA, PROCEDURE".

To memorize that easily, memorize this easy sentence: "I enjoy data processing". In that sentence, the words begin with the letters "I E D P"— and so do the four COBOL divisions.

In the program, each blank that you fill is called a *paragraph*. In the first paragraph, write the program's name; in the next paragraph, write your *own* name; in the next two paragraphs, write the computer's name; and in the last paragraph, write what you want the computer to do. The first paragraph is called the PROGRAM-ID; the next paragraph is called the AUTHOR; the next two paragraphs are called the SOURCE-COMPUTER and the OBJECT-COMPUTER; and the last paragraph is called the MAIN-ROUTINE.

Each paragraph is indented. To indent on PDP and Eclipse computers, type a controlled I.

In COBOL, the only important punctuation mark is the period. When writing a simple program, put a period at the end of each line. COBOL never requires commas or semicolons.

Don't forget the hyphens! Put a hyphen in PROGRAM-ID, SOURCE-COMPUTER, OBJECT-COMPUTER, and MAIN-ROUTINE.

Use correct spacing:

Right	Wrong	Wrong	Wrong
DISPLAY "BURP".	DIS PLAY "BURP".	DISPLAY"BURP".	DISPLAY "BURP" .

In the paragraphs that are called SOURCE-COMPUTER and OBJECT-COMPUTER, you must type the computer's name correctly. Here are the correct names for some famous computers:

```
IBM-360.
IBM-370.
PDP-11.
DECSYSTEM-10.    (It means you have a PDP-10.)
DECSYSTEM-20.    (It means you have a PDP-20.)
ECLIPSE C300.
6600.            (It means you have a CDC 6600.)
```

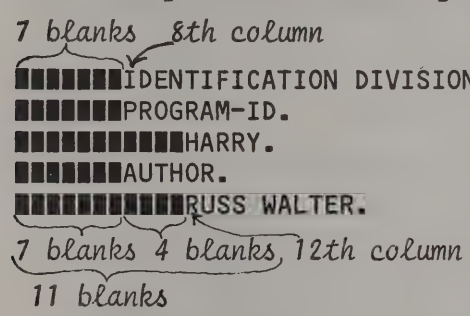
Don't forget the hyphens!

Old-fashioned computers

On IBM computers, instead of using quotation marks, you must use apostrophes:

Most computers	IBM computers
`DISPLAY "LIFE STINKS".`	`DISPLAY 'LIFE STINKS'.`

On IBM and CDC computers, you must indent the entire program, like this (each ▮ represents a blank space):



On those computers, each paragraph begins in column 12, and must not go farther to the right than column 72. (The computer ignores everything in columns 73-80.)

Try it

Go ahead: try writing your *own* COBOL program. In the PROCEDURE DIVISION, remember to say DISPLAY:

BASIC
```
10 PRINT "LIFE STINKS"
```

FORTRAN
```
      PRINT 10
10    FORMAT(1X,'LIFE STINKS')
```

PASCAL
```
WRITELN('LIFE STINKS');
```

COBOL
```
DISPLAY "LIFE STINKS".
```

ABRIDGMENTS

If you're lazy, you can omit the AUTHOR paragraph:

Complete IDENTIFICATION DIVISION

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
        HARRY.
AUTHOR.
        RUSS WALTER.
```

Abridged IDENTIFICATION DIVISION

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
        HARRY.
```

If you're lazy, *and you're using a PDP-10, PDP-20, Eclipse, or IBM computer*, you can omit the CONFIGURATION SECTION, SOURCE-COMPUTER, and OBJECT-COMPUTER:

Complete ENVIRONMENT DIVISION

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
        The computer's name.
OBJECT-COMPUTER.
        The computer's name.
```

Abridged ENVIRONMENT DIVISION

```
ENVIRONMENT DIVISION.
```

If you're *very* lazy, and you're using a PDP-10 or PDP-20 computer, you can abridge the program even further, so that the entire program looks like this:

```
IDENTIFICATION DIVISION.
PROCEDURE DIVISION.
        DISPLAY "LIFE STINKS".
        STOP RUN.
```

But if you're working for a big company, your employer will expect you to *not* be lazy: if you're lazy, you get fired!

FANCY DISPLAYS

You've seen that every COBOL program consists of four divisions: IDENTIFICATION, ENVIRONMENT, DATA, and PROCEDURE. The most important division is the PROCEDURE DIVISION. Let's look at it more closely.

Here's a cute PROCEDURE DIVISION:

```
PROCEDURE DIVISION.
MAIN-ROUTINE.
        DISPLAY "BILLIE AND BONNIE".
        DISPLAY "BURP".
        STOP RUN.
```

It makes the computer display:

```
BILLIE AND BONNIE
BURP
```

Another example:

```
PROCEDURE DIVISION.
MAIN-ROUTINE.
        DISPLAY "FLU" "SHED".
        STOP RUN.
```

The computer will display FLU and SHED on the same line:

```
FLUSHED
```

PERFORM

Let's make the computer display "I LOVE YOU", then display "I HATE YOU" six times, then display "I AM CONFUSED", like this:

```
I LOVE YOU
I HATE YOU
I HATE YOU
I HATE YOU
I HATE YOU
I HATE YOU
I HATE YOU
I AM CONFUSED
```

Here's the PROCEDURE DIVISION:

```
PROCEDURE DIVISION.
MAIN-ROUTINE.
        DISPLAY "I LOVE YOU".
        PERFORM EXPRESS-THE-HATRED 6 TIMES.
        DISPLAY "I AM CONFUSED.
        STOP RUN.
EXPRESS-THE-HATRED.
        DISPLAY "I HATE YOU".
```

The PROCEDURE DIVISION consists of two paragraphs: the first paragraph is called the MAIN-ROUTINE; the second paragraph is called EXPRESS-THE-HATRED.

The computer obeys the first paragraph: it displays "I LOVE YOU", then performs the EXPRESS-THE-HATRED 6 times, then displays "I AM CONFUSED", and finally stops.

When you invent your own PROCEDURE DIVISION, the first paragraph should be called the MAIN-ROUTINE; for the other paragraphs underneath, invent whatever names you like (such as EXPRESS-THE-HATRED). A paragraph's name should be hyphenated, and should contain no more than 30 characters. (EXPRESS-THE-HATRED contains 18 characters, so it's okay.) The first paragraph is the main routine, the paragraphs underneath are subroutines. The bottom line of the main routine should say STOP RUN. In the middle of the main routine, you should say to PERFORM the subroutines.

In the example above, the main routine says to PERFORM the EXPRESS-THE-HATRED subroutine 6 times. If you'd like to see more hatred, say 100 times instead of 6:

```
        PERFORM EXPRESS-THE-HATRED 100 TIMES.
```

If you'd rather see just a *little* hatred, say just—

```
        PERFORM EXPRESS-THE-HATRED 1 TIMES.
```

or say just:

```
        PERFORM EXPRESS-THE-HATRED.
```

Let's make the computer display:

```
I KNOW THAT
YOU ARE DRIVING
ME CRAZY
YOU ARE DRIVING
ME CRAZY
YOU ARE DRIVING
ME CRAZY
YOU ARE DRIVING
ME CRAZY
YOU ARE DRIVING
ME CRAZY
AND YET I LOVE YOU
```

Here's the PROCEDURE DIVISION:

```
PROCEDURE DIVISION.
MAIN-ROUTINE.
        DISPLAY "I KNOW THAT".
        PERFORM ACT-AS-IF-GOING-INSANE 5 TIMES.
        DISPLAY "AND YET I LOVE YOU".
        STOP RUN.
ACT-AS-IF-GOING-INSANE.
        DISPLAY "YOU ARE DRIVING".
        DISPLAY "ME CRAZY".
```

Let's make the computer display:

```
THE ASTRONAUTS GO
UP
UP
UP
UP
UP
UP
UP
AND THEN THEY COME
DOWN
DOWN
DOWN
DOWN
DOWN
DOWN
DOWN
```

Here's the PROCEDURE DIVISION:

```
PROCEDURE DIVISION.
MAIN-ROUTINE.
        DISPLAY "THE ASTRONAUTS GO".
        PERFORM SHOW-THE-ASTRONAUTS-RISING 7 TIMES.
        DISPLAY "AND THEN THEY COME".
        PERFORM SHOW-THE-ASTRONAUTS-FALLING 7 TIMES.
        STOP RUN.
SHOW-THE-ASTRONAUTS-RISING.
        DISPLAY "UP".
SHOW-THE-ASTRONAUTS-FALLING.
        DISPLAY "DOWN".
```

Let's make the computer display:

```
YOU ARE VERY SWEET
HA-HA-HA!
HO-HO-HO!
YOU CANNOT BE BEAT
HA-HA-HA!
HO-HO-HO!
YOUR LIPS ARE LIKE WINE
HA-HA-HA!
HO-HO-HO!
BUT YOU SMELL LIKE TURPENTINE
HA-HA-HA!
HO-HO-HO!
YOU STINK!
```

Here's the PROCEDURE DIVISION:

```
PROCEDURE DIVISION.
MAIN-ROUTINE.
        DISPLAY "YOU ARE VERY SWEET".
        PERFORM LAUGH-A-LOT.
        DISPLAY "YOU CANNOT BE BEAT".
        PERFORM LAUGH-A-LOT.
        DISPLAY "YOUR LIPS ARE LIKE WINE".
        PERFORM LAUGH-A-LOT.
        DISPLAY "BUT YOU SMELL LIKE TURPENTINE".
        PERFORM LAUGH-A-LOT.
        DISPLAY "YOU STINK!".
        STOP RUN.
LAUGH-A-LOT.
        DISPLAY "HA-HA-HA!".
        DISPLAY "HO-HO-HO!".
```

Variables

STRING VARIABLES

Like BASIC, FORTRAN, and PASCAL, COBOL lets you use variables. The name of a variable can be a letter (like X or Y) or a hyphenated phrase (like NUMBER-OF-BULLIES-I-SQUIRTED). A hyphenated phrase can have up to 30 characters.

To use a variable, you must describe it in the data division, as in this example:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
        JUNK.
AUTHOR.
        RUSS WALTER.          Instead of "RUSS WALTER", write your own name.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
        DECSYSTEM-20.         Instead of "DECSYSTEM-20", write your own computer's name.
OBJECT-COMPUTER.
        DECSYSTEM-20.

DATA DIVISION.
WORKING-STORAGE SECTION.      In the DATA DIVISION, you must say WORKING-STORAGE SECTION.
01      K PICTURE IS XXX.

PROCEDURE DIVISION.
MAIN-ROUTINE.
        MOVE "HER" TO K.
        DISPLAY "PUS" K "S".
        STOP RUN.
```

In the DATA DIVISION's WORKING-STORAGE SECTION, the "01 K" says K is a variable. The PICTURE IS XXX says K is a string that has three characters; each X stands for a character. In the PROCEDURE DIVISION, the first sentence makes K become this 3-character string: "HER". The next sentence makes the computer display:

```
PUSHERS
```

When you type that program, make sure you put a hyphen between WORKING and STORAGE. If you forget the hyphen, the computer will act crazy, and will say that your program contains many, many errors.

Suppose you change the picture from XXX to XX, so that the DATA DIVISION and PROCEDURE DIVISION look like this:

```
  DATA DIVISION.
  WORKING-STORAGE SECTION.
↠01      K PICTURE IS XX.

  PROCEDURE DIVISION.
  MAIN-ROUTINE.
          MOVE "HER" TO K.
          DISPLAY "PUS" K "S".
          STOP RUN.
```

K will be a string having only two characters. When the computer tries to move "HER" to K, only the first two characters of "HER" will fit, so K will be "HE". The computer will display:

```
PUSHES
```

Suppose you change the picture to XXXX. K will have four characters. When the computer tries to move "HER" to K, it needs to move a fourth character also, so it moves a blank space at the end, which makes K be "HER ". The computer will display:

```
PUSHER S
```

This program shows how revolutionary politics leads to revolutionary clothing:

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01      FIRST-PRESIDENT PICTURE IS XXXXXXXXXX.
01      CLEANING-METHOD PICTURE IS XXXX.

PROCEDURE DIVISION.
MAIN-ROUTINE.
        MOVE "WASHINGTON" TO FIRST-PRESIDENT.
        MOVE FIRST-PRESIDENT TO CLEANING-METHOD.
        DISPLAY CLEANING-METHOD " MY BLUE JEANS".
        STOP RUN.
```

Above the DATA DIVISION, write your own IDENTIFICATION DIVISION and ENVIRONMENT DIVISION. The DATA DIVISION says FIRST-PRESIDENT will be a string having ten characters, and CLEANING-METHOD will be a string having four. The first sentence of the MAIN-ROUTINE makes FIRST-PRESIDENT be "WASHINGTON". The next sentence tries to move "WASHINGTON" to CLEANING-METHOD; but because of CLEANING-METHOD's picture, the computer moves "WASH" instead. The computer will display:

```
WASH MY BLUE JEANS
```

To make sure you understand the word MOVE, examine this example:

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01      C PICTURE IS XXX.
01      D PICTURE IS XXX.

PROCEDURE DIVISION.
MAIN-ROUTINE.
        MOVE "CAT" TO C.     C becomes "CAT".
        MOVE C TO D.         D becomes "CAT".
        DISPLAY C.           Since C is still "CAT", the computer displays "CAT".
        MOVE "HE" TO C.      C becomes "HE ".
        DISPLAY C "BLED".    The computer displays "HE BLED".
        STOP RUN.
```

COBOL allows abbreviations. You can say PIC instead of PICTURE IS, and X(7) instead of XXXXXXX.

NUMERIC VARIABLES

Let's make the computer add 53 and 4, and display the sum, 57. Here's how:

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01      K PIC 99.

PROCEDURE DIVISION.
MAIN-ROUTINE.
        COMPUTE K = 53 + 4.
        DISPLAY K.
        STOP RUN.
```

The PIC 99 says K is a number having two digits. (Each 9 stands for a digit.) The MAIN-ROUTINE sets K equal to 53 + 4, which is 57. The computer will display:

```
57
```

In the COMPUTE statement, the equal sign and the plus sign must be surrounded by spaces:

Right	Wrong	Wrong
`COMPUTE K = 53 + 4.`	`COMPUTE K=53+4.`	`COMPUTE K = 53+4.`

If you change the picture to 999, K will be a number having three digits. It will be 057 instead of 57. The computer will display:

```
057
```

(Exception: PDP-10 and PDP-20 computers are lazy; they don't bother to display the 0 at the left; they display just 57.)

If you change the picture to 9, K will be a number having only one digit. So K will not be 57. The computer will not display the correct sum.

Like FORTRAN, COBOL uses these operators:

Operator	Meaning
+	plus
-	minus
*	times
/	divided by
**	exponent

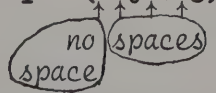
You must put a blank space before and after each operator:

Right	Wrong
`53 + 4`	`53+4`
`7 ** 2`	`7**2`
`- J + 3`	`-J + 3`

Like BASIC, FORTRAN, and PASCAL, COBOL lets you use parentheses. Do *not* put a space after a left parenthesis:

Right

```
COMPUTE K = I * (- J + 3)
```



You can use these short cuts:

Sentence	Short cut
`COMPUTE A = A + 7.`	`ADD 7 TO A.`
`COMPUTE B = B - 4.`	`SUBTRACT 4 FROM B.`

Operators are allowed only in sentences that say COMPUTE, IF, UNTIL, or WHEN:

Allowed	Not allowed	Not allowed	Not allowed
`COMPUTE A = 2 * 3.`	`DISPLAY 2 * 3.`	`MOVE 2 * 3 TO A.`	`ADD 2 * 3 TO A.`

DECIMALS

You can use decimals:

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01      K PIC 9999V99.

PROCEDURE DIVISION.
MAIN-ROUTINE.
        COMPUTE K = 2.208 + 4.109.
        DISPLAY K.
        STOP RUN.
```

The PIC 9999V99 says K is a number having four digits, followed by a decimal point, followed by two digits. (The V stands for the decimal point.) The MAIN-ROUTINE tries to set K equal to 2.208 + 4.109, which is 6.317; but because of K's picture, I will be 0006.31 instead. So the computer should display 0006.31.

PDP-10 and PDP-20 computers don't bother to display the zeros: they display 6.31. PDP-11, IBM, CDC, and Eclipse computers don't bother to display the decimal point: they display 000631.

You can change that program by saying ROUNDED:

```
        COMPUTE K ROUNDED = 2.208 + 4.109.
```

The computer will find the sum (6.317), and round it so K is 0006.32 instead of 0006.31.

If J's picture is 99, saying "COMPUTE J = 200 / 3" makes J be 66. Saying "COMPUTE J ROUNDED = 200 / 3" makes J be 67.

Which is better: saying "COMPUTE X = 4.9" or "MOVE 4.9 TO X"? You should usually say "MOVE 4.9 TO X", because the computer handles it more quickly. But MOVE cannot round; so if you want to round, say COMPUTE.

MOVE can do strange things:

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01      L PIC 999V99.

PROCEDURE DIVISION.
MAIN-ROUTINE.
        MOVE 16725.048 TO L.
        DISPLAY L.
        STOP RUN.
```

L's picture makes the computer move the three digits just left of the decimal point and the two digits just right of it. L will be 725.04.

NEGATIVES

You can use negative numbers:

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01      K PIC S9999.

PROCEDURE DIVISION.
MAIN-ROUTINE.
        COMPUTE K = 100 - 367.
        DISPLAY K.
        STOP RUN.
```

In K's picture, the S stands for a sign (which can be plus or minus). K will be -0267. The computer should display -0267.

PDP-10 and PDP-20 computers don't bother to display the zero: they display -267. Some other computers display the minus sign on top of the right digit, like this: 0267. On most computers, the minus combines with the 7 and forms a P, like this: 026P.

If you omit the S from K's picture, K will be 0267 instead of -0267.

To find the negative of a power, use parentheses:

```
COMPUTE A = - (3 ** 2).
```

If you omit the parentheses, the computer will get the wrong answer.

ACCEPT

Here's how to translate the BASIC word *INPUT* into FORTRAN, PASCAL, and COBOL:

BASIC	FORTRAN	PASCAL	COBOL
INPUT K	READ *, K	READ(K);	ACCEPT K.

Let's look at a COBOL example:

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01      K PIC XXX.

PROCEDURE DIVISION.
MAIN-ROUTINE.
        DISPLAY "THIS PROGRAM WANTS YOU TO TYPE SOMETHING".
        ACCEPT K.
        DISPLAY K.
        STOP RUN.
```

The computer displays:

```
THIS PROGRAM WANTS YOU TO TYPE SOMETHING
```

The statement ACCEPT K makes the computer wait for you to type something. If you type—

```
FIGHT
```

the computer will try to move "FIGHT" to K; but since K's picture is XXX, K will be "FIG". The computer will display:

```
FIG
```

If you type—

```
ME
```

the computer will try to move "ME" to K; since K's picture is XXX, K will be "ME ". The computer will display:

```
ME
```

Suppose a program says L PIC 999 and ACCEPT L. If you input—

```
4
```

a PDP-10 or PDP-20 computer will make L be 004, but an IBM or CDC computer will make L be 400.

Suppose a program says M PIC S9999V99 and ACCEPT M. If you want M to be -0034.27, here's what to input:

On PDP-10 and PDP-20 computers	On IBM and CDC computers
-0034.27 *or* -34.27	003427

EDITING

The computer can edit the output:

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01      K PIC XXXBXXX.

PROCEDURE DIVISION.
MAIN-ROUTINE.
        MOVE "HITHER" TO K.
        DISPLAY K.
        STOP RUN.
```

In K's picture, B means a blank space. So when the computer moves "HITHER", K becomes "HIT HER". The computer will display:

```
HIT HER
```

This program displays my phone number:

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01      PHONE-NUMBER PIC 999B9999.

PROCEDURE DIVISION.
MAIN-ROUTINE.
        MOVE 2668128 TO PHONE-NUMBER.
        DISPLAY PHONE-NUMBER.
        STOP RUN.
```

When the computer moves 2668128, PHONE-NUMBER becomes "266 8128". The computer will display:

```
266 8128
```

This program is of historical importance:

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01      K PIC 9B9B9999.

PROCEDURE DIVISION.
MAIN-ROUTINE.
        COMPUTE K = 741775 + 1.
        DISPLAY "THE DECLARATION OF INDEPENDENCE WAS SIGNED ON " K.
        STOP RUN.
```

The computer will display:

```
THE DECLARATION OF INDEPENDENCE WAS SIGNED ON 7 4 1776
```

Using B to insert a blank is called *editing*. You've learned about four kinds of variables:

Kind of variable	Symbols in picture
string	X
edited string	X B
number	9 V S
edited number	9 B Z , $ * . + - DB CR

You can use these pictures for editing numbers:

K's picture	Meaning	What K will be if you move 50320 to K	What K will be if you move 0 to K
B99999999	a blank followed by eight digits	" 00050320"	" 00000000"
ZZZZZZZZZ	blanks then digits	" 50320"	" "
Z,ZZZ,ZZZ	insert commas where appropriate	" 50,320"	" "
$,$$$,$$$	put a dollar sign before the digits	" $50,320"	" "
*,***,***	put asterisks instead of blanks	"***50,320"	"*********"

To edit decimals, put a decimal point in the picture.

K's picture	If you move 50320.6 to K	If you move .04 to K	If you move 0 to K
Z,ZZZ,ZZZ.ZZ	" 50,320.60"	" .04"	" "
$,$$$,$$$.$$	" $50,320.60"	" $.04"	" "
*,***,***.**	"***50,320.60"	"*********.04"	"*********.**'

With those pictures, if you move 0 to K the computer doesn't put any digits in K. To guarantee that K contains a digit, put 9 in the picture:

K's picture	If you move 50320.6 to K	If you move .04 to K	If you move 0 to K
Z,ZZZ,ZZ9.99	" 50,320.60"	" 0.04"	" 0.00"
$,$$$,$$9.99	" $50,320.60"	" $0.04"	" $0.00"
*,***,**9.99	"***50,320.60"	"********0.04"	"********0.00"

To edit negative numbers, use +, -, DB, or CR.

K's picture	Meaning	If you move -2.6 to K	If move 2.6 to K
ZZZ.ZZ+	put - or + afterwards	" 2.60-"	" 2.60+"
ZZZ.ZZ-	put - or blank afterwards	" 2.60-"	" 2.60 "
ZZZ.ZZDB	if negative, put DB (for *debit*)	" 2.60DB"	" 2.60 "
ZZZ.ZZCR	if negative, put CR (for *credit*)	" 2.60CR"	" 2.60 "

For fancier pictures, replace the Z by $, *, or 9.

Here's how to put the sign before the digits:

K's picture	Meaning	If you move -2.6 to K	If move 2.6 to K
+++.++	put - or + before the digits	" -2.60"	" +2.60"
---.--	put - or blank before the digits	" -2.60"	" 2.60"

Here are the differences between numbers and edited numbers:

	Number	Edited number
how to put a decimal point in the picture	V	.
how to put a sign in the picture	S	+, -, DB, or CR
how to fill up most of the picture	9	9, Z, $, *, +, or -
what the value should be	an intermediate result in a long calculation	the final answer to be displayed
If the variable is called K, can you say ACCEPT K?	yes	no
If the variable is called K, can you use K in further computations? For example, can you say COMPUTE L = K + 1?	yes	no
What happens if you try to DISPLAY the value?	might display the zeros, decimal point, and minus sign incorrectly	displays correctly

Logic

IF

Like BASIC, FORTRAN, and PASCAL, COBOL uses the words IF and ELSE.

Newest BASIC:

```
INPUT I
IF I>5 THEN J=80: K=90
```

```
INPUT I
IF I>5 THEN J=80: K=90 ELSE J=30: K=50
```

FORTRAN 77:

```
READ *, I
IF (I .GT. 5) THEN
   J=80
   K=90
END IF
```

```
READ *, I
IF (I .GT. 5) THEN
   J=80
   K=90
ELSE
   J=30
   K=50
END IF
```

PASCAL:

```
READ(I);
IF I>5 THEN
        BEGIN;
        J:=80;
        K:=90;
        END;
```

```
READ(I);
IF I>5 THEN
        BEGIN;
        J:=80;
        K:=90;
        END
ELSE
        BEGIN;
        J:=30;
        K:=50;
        END;
```

COBOL:

```
ACCEPT I.
IF I > 5
        MOVE 80 TO J
        MOVE 90 TO K.
```

```
ACCEPT I.
IF I > 5
        MOVE 80 TO J
        MOVE 90 TO K
ELSE
        MOVE 30 TO J
        MOVE 50 TO K.
```

In COBOL, when you write the IF statement, do *not* put a period at the end of every line; instead, put the period just at the end of the entire IF idea.

Notice that I indented the word MOVE. The indentation is optional, but is a good habit. To indent on PDP and Eclipse computers, type a controlled I; to indent on IBM and CDC computers, press the space bar several times.

COBOL uses these IF lines:

IF line	Meaning
`IF I = 5`	If I is equal to 5
`IF I NOT = 5`	If I is not equal to 5
`IF I > 5`	If I is greater than 5
`IF I NOT > 5`	If I is not greater than 5
`IF I < 5`	If I is less than 5
`IF I NOT < 5`	If I is not less than 5

You can use the words AND and OR and abbreviate:

IF line	Abbreviation
`IF J > 1 AND J < 100`	`IF J > 1 AND < 100`
`IF K < -3 OR K = 6 OR K = 9 OR K = 12 OR K > 50`	`IF K < -3 OR = 6 OR 9 OR 12 OR > 50`

You can say this:

```
IF AGE < 13
        DISPLAY "CHILD"
ELSE IF AGE < 20
        DISPLAY "TEENAGER"
ELSE IF AGE < 40
        DISPLAY "YOUNG ADULT"
ELSE IF AGE < 60
        DISPLAY "MIDDLE-AGED"
ELSE
        DISPLAY "SENIOR CITIZEN".
```

It means, "If AGE is less than 13, display the word CHILD; if not less than 13, do the following: if AGE is less than 20, display the word TEENAGER; if not less than 20, do the following: if AGE is less than 40,..." and so on. The computer will display just one phrase, to describe the person's AGE.

UNTIL

You can say UNTIL:

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01      I PIC 999.

PROCEDURE DIVISION.
MAIN-ROUTINE.
        MOVE 5 TO I.
        PERFORM FIDDLE-WITH-I UNTIL I > 100.
        STOP RUN.
FIDDLE-WITH-I.
        DISPLAY I.
        COMPUTE I = I * 2.
```

The computer will perform FIDDLE-WITH-I repeatedly, until I > 100. The computer will display 5, 10, 20, 40, and 80. It will not display 160.

Here are the details. When the computer encounters PERFORM FIDDLE-WITH-I UNTIL I > 100, it checks whether I > 100. If I > 100, the computer proceeds to the next statement (the STOP RUN); but if I is *not* greater than 100, the computer performs FIDDLE-WITH-I and then re-executes the statement PERFORM FIDDLE-WITH-I UNTIL I > 100.

To translate a BASIC "FOR...NEXT loop" into COBOL, say "PERFORM":

BASIC

```
10 FOR I = 5 TO 17
20     PRINT I
30 NEXT I
```

COBOL

```
PROCEDURE DIVISION.
MAIN-ROUTINE.
        PERFORM DISPLAY-IT
                VARYING I FROM 5 BY 1 UNTIL I > 17.
        STOP RUN.
DISPLAY-IT.
        DISPLAY I.
```

BASIC

```
10 FOR I = 5 TO 17 STEP 3
20     PRINT I
30 NEXT I
```

COBOL

```
PROCEDURE DIVISION.
MAIN-ROUTINE.
        PERFORM DISPLAY-IT
                VARYING I FROM 5 BY 3 UNTIL I > 17.
        STOP RUN.
DISPLAY-IT.
        DISPLAY I.
```

BASIC

```
10 FOR I = 5 TO 17
20     FOR J = 1 TO 3
30         PRINT I,J
40     NEXT J
50 NEXT I
```

COBOL

```
PROCEDURE DIVISION.
MAIN-ROUTINE.
        PERFORM DISPLAY-IT
                VARYING I FROM 5 BY 1 UNTIL I > 17
                AFTER VARYING J FROM 1 BY 1 UNTIL J > 3.
        STOP RUN.
DISPLAY-IT.
        DISPLAY I J.
```

GO TO

Like BASIC, FORTRAN, and PASCAL, COBOL lets you say GO TO. In COBOL, put a space between GO and TO. (In PASCAL, you do *not* put a space between GO and TO.)

For example, instead of saying STOP RUN, you can say GO TO MAIN-ROUTINE:

```
PROCEDURE DIVISION.
MAIN-ROUTINE.
        DISPLAY "WHEATIES".
        DISPLAY "ARE WONDERFUL".
        GO TO MAIN-ROUTINE.
```

The computer will display "WHEATIES" and "ARE WONDERFUL", repeatedly:

```
WHEATIES
ARE WONDERFUL
WHEATIES
ARE WONDERFUL
WHEATIES
ARE WONDERFUL
etc.
```

The main routine can consist of *two* paragraphs, called MAIN-ROUTINE-BEGINNING and MAIN-ROUTINE-LOOP:

```
PROCEDURE DIVISION.
MAIN-ROUTINE-BEGINNING.
        DISPLAY "PLEASE".
MAIN-ROUTINE-LOOP.
        DISPLAY "KISS".
        DISPLAY "ME".
        GO TO MAIN-ROUTINE-LOOP.
```

The computer will display PLEASE, then repeatedly display KISS and ME:

```
PLEASE
KISS
ME
KISS
ME
KISS
ME
etc.
```

The main routine can consist of *three* paragraphs, called MAIN-ROUTINE-BEGINNING, MAIN-ROUTINE-LOOP, and MAIN-ROUTINE-ENDING:

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01      HUMAN-RESPONSE PIC XXX.

PROCEDURE DIVISION.
MAIN-ROUTINE-BEGINNING.
        DISPLAY "I WILL RECITE A SHORT POEM".
MAIN-ROUTINE-LOOP.
        DISPLAY " ".
        DISPLAY "YOUR NOSE".
        DISPLAY "BLOWS".
        DISPLAY " ".
        DISPLAY "WOULD YOU LIKE TO HEAR THE POEM AGAIN?".
        ACCEPT HUMAN-RESPONSE.
        IF HUMAN-RESPONSE = "YES"
                GO TO MAIN-ROUTINE-LOOP.
MAIN-ROUTINE-ENDING.
        DISPLAY "YOU HAVE BEEN A GREAT AUDIENCE".
        STOP RUN.
```

When you run that program, the computer says:

```
I WILL RECITE A SHORT POEM
```

Then it recites the program:

```
YOUR NOSE
BLOWS
```

Then it asks:

```
WOULD YOU LIKE TO HEAR THE POEM AGAIN?
```

If you say YES, the computer repeats the poem, and then asks you whether you'd like to hear it a third time. If you say YES again, the computer recites the poem a third time, and then asks whether you'd like to hear it a *fourth* time. The computer recites the poem repeatedly, until you finally stop saying YES. Then the computer says—

```
YOU HAVE BEEN A GREAT AUDIENCE
```

and stops.

In that program, if you don't say YES, the computer doesn't repeat the poem. So if you don't say YES, the computer acts as if you said NO. The following version is an improvement; if you don't say YES, and you don't say NO, the computer asks the question again:

```
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  01      HUMAN-RESPONSE PIC XXX.

  PROCEDURE DIVISION.
  MAIN-ROUTINE-BEGINNING.
          DISPLAY "I WILL RECITE A SHORT POEM".
  MAIN-ROUTINE-LOOP.
          DISPLAY " ".
          DISPLAY "YOUR NOSE".
          DISPLAY "BLOWS".
          DISPLAY " ".
→→        PERFORM GET-HUMAN-RESPONSE.
          IF HUMAN-RESPONSE = "YES"
                  GO TO MAIN-ROUTINE-LOOP.
  MAIN-ROUTINE-ENDING.
          DISPLAY "YOU HAVE BEEN A GREAT AUDIENCE".
          STOP RUN.
→→GET-HUMAN-RESPONSE.
→→        DISPLAY "WOULD YOU LIKE TO HEAR THE POEM AGAIN?".
→→        ACCEPT HUMAN-RESPONSE.
→→        IF HUMAN-RESPONSE NOT = "YES" AND NOT = "NO"
→→                DISPLAY "PLEASE SAY YES OR NO!"
→→                GO TO GET-HUMAN-RESPONSE.
```

GO TO resembles PERFORM. Here's the difference between GO TO and PERFORM....

To go to a *different routine*, say *PERFORM*. To go to a different paragraph in the *same* routine, say *GO TO*.

For example, suppose you want to go from MAIN-ROUTINE-BEGINNING to FUNNY-SUBROUTINE; since you're going to a *different routine*, say *PERFORM*.

Suppose you want to go from MAIN-ROUTINE-BEGINNING to MAIN-ROUTINE-ENDING; since you're going to a different paragraph in the *same* routine, say *GO TO*.

THRU

Like the main routine, a subroutine can consist of several paragraphs. For example, subroutine FUNNY-FACE can consist of three paragraphs, called FUNNY-FACE-BEGINNING, FUNNY-FACE-LOOP, and FUNNY-FACE-ENDING. To make the computer do the entire subroutine, say:

```
PERFORM FUNNY-FACE-BEGINNING THRU FUNNY-FACE-ENDING.
```

Structures

FILES

To manipulate a file whose name is POEM, fill in the blanks:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
        The program's name.
AUTHOR.
        Your name.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
        The computer's name.
OBJECT-COMPUTER.
        The computer's name again.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT POEM-FILE ASSIGN TO the file's location.

DATA DIVISION.
FILE SECTION.
FD      POEM-FILE how the file is labeled.
01      POEM-LINE PIC a picture of a line of the file.
WORKING-STORAGE SECTION.
A description of each variable that's not in the file.

PROCEDURE DIVISION.
MAIN-ROUTINE.
        OPEN output or input POEM-FILE.
        What you want to do to the file.
        CLOSE POEM-FILE.
        STOP RUN.
```

Like every COBOL program, that outline consists of four divisions: the IDENTIFICATION DIVISION, the ENVIRONMENT DIVISION, the DATA DIVISION, and the PROCEDURE DIVISION. Let's look at each division.

Identification division

The IDENTIFICATION DIVISION consists of two paragraphs: the PROGRAM-ID and the AUTHOR. For the PROGRAM-ID, fill in the program's name, *which must be different than the name of the file*. Since the name of the file is POEM, the name of the program must *not* be POEM. If you're lazy, you can omit the AUTHOR.

Environment division

The ENVIRONMENT DIVISION consists of two sections: the CONFIGURATION SECTION and the INPUT-OUTPUT SECTION.

The CONFIGURATION SECTION consists of two paragraphs: the SOURCE-COMPUTER and the OBJECT-COMPUTER. On CDC and PDP-11 computers, the CONFIGURATION SECTION is required; but on IBM, Eclipse, PDP-10, and PDP-20 computers, the entire CONFIGURATION SECTION is optional, so you can abridge the ENVIRONMENT DIVISION:

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT POEM-FILE ASSIGN TO the file's location.
```

The INPUT-OUTPUT SECTION consists of just one paragraph, which is the FILE-CONTROL. The FILE-CONTROL paragraph consists of a sentence that says SELECT, then the file's name (POEM-FILE), then ASSIGN TO, and finally a blank (which you must fill in, and which tells the file's location).

What do you put in that blank? The answer depends on the file's location. Is the file on a disk? On punched cards? Or on paper produced by the printer? Here's what to put in the blank, for various computers:

	Printer	Card reader	Disk
Eclipse	PRINTER	"$CDR"	"POEM"
CDC	OUTPUT	INPUT	POEM
PDP-11	"LP:"	"CR:"	"DK:"
PDP-10, PDP-20	LPT	CDR	DSK RECORDING MODE ASCII
IBM using OS	UT-S-POEM	UT-S-POEM	UT-S-POEM
IBM using DOS	SYS006-UR-1403-S	SYS005-UR-2540R-S	SYS020-UT-3330-S-POEM

Data division

The DATA DIVISION consists of two sections: the FILE SECTION and the WORKING-STORAGE SECTION.

At the beginning of the FILE SECTION, say FD (which means "File Description"). To the right of the FD, say POEM-FILE, and then fill in the blank, which tells how the file is labeled:

Computer	What to put in the blank
CDC	LABEL RECORDS ARE OMITTED
PDP-11: disk	LABEL RECORDS ARE STANDARD VALUE OF ID "POEM"
PDP-11: printer or card reader	LABEL RECORDS ARE OMITTED
PDP-10, PDP-20	VALUE ID "POEM␣␣␣␣␣"
IBM OS	LABEL RECORDS ARE STANDARD
IBM DOS: disk	LABEL RECORDS ARE STANDARD
IBM DOS: printer or card reader	LABEL RECORDS ARE OMITTED

On PDP-10 and PDP-20 computers, put enough blank spaces (␣) after POEM so that the string has 9 characters. On Eclipse computers, do *not* fill in the blank; just say:

```
FD      POEM-FILE.
```

Underneath the line that says FD, you must say 01. The 01 line includes a picture of a line of the file. For example, if a line of the file is an 80-character string, the 01 line should say:

```
01      POEM-LINE PIC X(80).
```

The WORKING-STORAGE SECTION describes each variable that's not in the file.

Procedure division

The PROCEDURE DIVISION's MAIN-ROUTINE should begin with the word OPEN, and end with the words CLOSE and STOP RUN. In the OPEN statement, you can say either—

```
        OPEN OUTPUT POEM-FILE.
```

or:

```
        OPEN INPUT POEM-FILE.
```

If you say OPEN OUTPUT POEM-FILE, the computer will output to the POEM-FILE; so it will copy information from the RAM to the POEM-FILE. If you say OPEN INPUT POEM-FILE, the computer will input from the POEM-FILE; so it will copy information from the POEM-FILE to the RAM.

In the PROCEDURE DIVISION, when you fill in the blank about "what you want to do to the file", you must say either WRITE POEM-LINE or READ POEM-FILE. If the file is OPEN OUTPUT (which means you're copying from the RAM to the file), say WRITE POEM-LINE; if the file is OPEN INPUT (which means you're copying from the file to the RAM), say READ POEM-FILE.

Writing

Here's a poetic masterpiece:

```
CANDY IS DANDY
BUT LIKKER IS QUIKKER
```

It was composed by the famous poet Ogden Nash. Let's make the computer write that masterpiece onto a disk, and make the masterpiece become a file named POEM.

Here's how:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
        CANDY.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
        The computer's name.
OBJECT-COMPUTER.
        The computer's name again.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT POEM-FILE ASSIGN TO the file's location.

DATA DIVISION.
FILE SECTION.
FD      POEM-FILE how the file is labeled.
01      POEM-LINE PIC X(21).     Therefore, each line will have 21 characters, like this:
                                 CANDY IS DANDY■■■■■■■
PROCEDURE DIVISION.              BUT LIKKER IS QUIKKER
MAIN-ROUTINE.
        OPEN OUTPUT POEM-FILE.                  The computer creates POEM-FILE for output.
        MOVE "CANDY IS DANDY" TO POEM-LINE.     POEM-LINE is "CANDY IS DANDY■■■■■■■".
        WRITE POEM-LINE.                        That POEM-LINE is copied onto the file.
        MOVE "BUT LIKKER IS QUIKKER" TO POEM-LINE.  POEM-LINE becomes a new string.
        WRITE POEM-LINE.                        That new POEM-LINE is copied onto the file.
        CLOSE POEM-LINE.
        STOP RUN.
```

That program doesn't require a WORKING-STORAGE SECTION, so I omitted it. Since I was lazy, I also omitted the AUTHOR paragraph. When you run that program, the computer will create a file on disk; the file will be called POEM; it will contain this message:

```
CANDY IS DANDY■■■■■■■
BUT LIKKER IS QUIKKER
```

Reading

This program reads the file POEM, and displays it on your terminal:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
        READER.
```

The ENVIRONMENT DIVISION and DATA DIVISION are the same as the previous program's.

```
PROCEDURE DIVISION.
MAIN-ROUTINE-BEGINNING.
        OPEN INPUT POEM-FILE.        Find POEM on the disk, & prepare to input from it.
MAIN-ROUTINE-LOOP.
        READ POEM-FILE AT END GO TO MAIN-ROUTINE-ENDING.  Read a line from POEM-FILE.
        DISPLAY POEM-LINE.           Display that line, so you see it on your terminal.
        GO TO MAIN-ROUTINE-LOOP.     Go back to read another line.
MAIN-ROUTINE-ENDING.
        DISPLAY "THAT WAS THE WHOLE POEM".
        CLOSE POEM-FILE.
        STOP RUN.
```

In the MAIN-ROUTINE-LOOP, the first line means: try to READ a line from POEM-FILE; but if a line can*not* be read (because the file has ended), go to MAIN-ROUTINE-ENDING instead.

The READ statement differs from the WRITE statement in two ways:

A WRITE statement mentions a LINE, but a READ statement mentions a FILE.
A READ statement must contain the words AT END.

Counting

This program reads a file called POEM, counts how many lines are in it, and displays the count:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
        COUNTS.
```

The ENVIRONMENT DIVISION is the same as the previous program's.

```
DATA DIVISION.
FILE SECTION.
FD      POEM-FILE how the file is labeled.
01      POEM-LINE PIC X(21).                    assumes POEM-LINE has 21 characters
WORKING-STORAGE SECTION.
01      COUNT-OF-HOW-MANY-LINES PIC 99.         assumes the count is a two-digit number;
                                                so assumes POEM has less than 100 lines

PROCEDURE DIVISION.
MAIN-ROUTINE-BEGINNING.
        OPEN INPUT POEM-FILE.                   Find POEM on the disk.
        MOVE 0 TO COUNT-OF-HOW-MANY-LINES.      Start the count at 0.
MAIN-ROUTINE-LOOP.
        READ POEM-FILE AT END GO TO MAIN-ROUTINE-ENDING.  Read a line from POEM-FILE.
        ADD 1 TO COUNT-OF-HOW-MANY-LINES.       Add 1 to the count.
        GO TO MAIN-ROUTINE-LOOP.                Go to read another line.
MAIN-ROUTINE-ENDING.                            When the file is done,
        DISPLAY COUNT-OF-HOW-MANY-LINES.        display the count.
        CLOSE POEM-FILE.
        STOP RUN.
```

Copying

This program reads a file called POEM, and creates a copy of it; the copy is a file called POEM2:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
        COPIER.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
        The computer's name.
OBJECT-COMPUTER.
        The computer's name again.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT POEM-FILE ASSIGN TO the location of POEM.
        SELECT POEM2-FILE ASSIGN TO the location of POEM-2.

DATA DIVISION.
FILE SECTION.
FD      POEM-FILE the labeling for POEM.
01      POEM-LINE PIC X(21).
FD      POEM-FILE the labeling for POEM2.
01      POEM2-LINE PIC X(21).

PROCEDURE DIVISION.
MAIN-ROUTINE-BEGINNING.
        OPEN INPUT POEM-FILE.          The computer will input from POEM-FILE,
        OPEN OUTPUT POEM2-FILE.        and output to POEM2-FILE.
MAIN-ROUTINE-LOOP.                     The computer does the following repeatedly:
        READ POEM-FILE AT END GO TO MAIN-ROUTINE-ENDING.  read a line from POEM-FILE;
        MOVE POEM-LINE TO POEM2-LINE.  copy that line to POEM2-LINE;
        WRITE POEM2-LINE.              and write POEM2-LINE onto POEM2-FILE.
        GO TO MAIN-ROUTINE-LOOP.
MAIN-ROUTINE-ENDING.
        DISPLAY "THE FILE HAS BEEN COPIED".
        CLOSE POEM-FILE POEM2-FILE.
        STOP RUN.
```

Pictures

Suppose you're dealing with a file named JOE, and each line of JOE-FILE is a three-digit number. Should the line's picture be edited (JOE-LINE PIC ZZZ) or unedited (JOE-LINE PIC 999)?

When you read a file, the line's picture must be unedited and match the picture in the program that wrote the file.

When you write a file, ask yourself, "What will read it?" If the answer is "a COBOL program", the picture must be unedited. If the answer is "only a human", edit the picture.

Remember: if one program writes a file, and another program reads it, both programs must use the same picture. For example, if a program writes JACK-FILE and says JACK-LINE PIC S9999V99, the program that reads JACK-FILE must also say JACK-LINE PIC S9999V99.

Peculiarities

To write and read unedited numbers, the computer takes a short-cut: it omits decimal points, and locates the negative sign on top of the last digit. For example, instead of writing -0034.27 in JACK-FILE, the computer writes just 003427. When another program reads 003427 from the file, the S9999V99 picture tells the computer the 003427 means -0034.27.

After you WRITE a line, you cannot use the line again in the program. For example, after you say WRITE POEM-LINE, you should not say MOVE POEM-LINE TO K; it won't work.

Multiple widths

Let's make the computer compute the square of 12 and the square of 13 and write this file:

```
HERE ARE THE SQUARES:
144
169
THEY WERE REAL GROOVY
```

The top and bottom lines are long strings whose pictures are X(21). The other two lines are short numbers whose pictures are 999. Here's the program:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
        SQUARE.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
        The computer's name.
OBJECT-COMPUTER.
        The computer's name again.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT REPORT-FILE ASSIGN TO the location of REPORT.  The file is named REPORT.

DATA DIVISION.
FILE SECTION.
FD      REPORT-FILE the labeling for REPORT.
01      REPORT-LINE PIC X(21).          REPORT-LINE is a 21-character string.
01      REPORT-LINE2 PIC 999.           REPORT-LINE2 is a 3-digit number.

PROCEDURE DIVISION.
MAIN-ROUTINE.
        OPEN OUTPUT REPORT-FILE.        The computer will create a file named REPORT.
        MOVE "HERE ARE THE SQUARES:" TO REPORT-LINE.
        WRITE REPORT-LINE.              The computer writes "HERE ARE THE SQUARES:".
        COMPUTE REPORT-LINE2 = 12 * 12. REPORT-LINE2 is 144.
        WRITE REPORT-LINE2.             The computer writes 144.
        COMPUTE REPORT-LINE2 = 13 * 13. REPORT-LINE2 is 169.
        WRITE REPORT-LINE2.             The computer writes 169.
        MOVE "THEY WERE REAL GROOVY" TO REPORT-LINE.
        WRITE REPORT-LINE.              The computer writes "THEY WERE REAL GROOVY".
        CLOSE REPORT-FILE.
        STOP RUN.
```

GROUP ITEMS

In the data division, you can say:

```
01      K.
        02      L PIC 999.
        02      M PIC 9.
        02      N PIC 99.
```

That means K is a combination of L, M, and N. If the procedure division says—

```
MOVE 427 TO L.
MOVE 8 TO M.
MOVE 31 TO N.
```

then K will be "427831".

Since K is a combination of other variables, K is called a *group variable* or *group item*. L, M, and N are *elementary items*. Notice that K is the string "427831", not the number 427831. A group item is always a string. Since K is a string, not a number, you cannot say ADD 1 TO K, although you can say ADD 1 TO L or ADD 1 TO M or ADD 1 TO N.

Here's a group item, for a weight-reducing studio:

```
01      PERSONAL-INFO-ABOUT-CLIENT.
        02      CLIENT-NAME.
                03      FIRST-NAME PIC X(15).
                03      MIDDLE-INITIAL PIC X.
                03      LAST-NAME PIC X(20).
        02      CLIENT-SEX PIC X.               A person's sex is "M" or "F".
        02      CLIENT-AGE PIC 99.
        02      WEIGHT-PROGRESS.
                03      WEIGHT-WHEN-ENTERED-PROGRAM PIC 999.
                03      WEIGHT-THIS-WEEK PIC 999.
                03      NUMBER-OF-WEEKS-SO-FAR PIC 999.
```

PERSONAL-INFO-ABOUT-CLIENT is composed of CLIENT-NAME (which is composed of FIRST-NAME, MIDDLE-INITIAL, and LAST-NAME), CLIENT-SEX, and WEIGHT-PROGRESS (which is composed of WEIGHT-WHEN-ENTERED-PROGRAM, WEIGHT-THIS-WEEK, and NUMBER-OF-WEEKS-SO-FAR). So PERSONAL-INFO-ABOUT-CLIENT is composed of numbers and strings. Altogether, PERSONAL-INFO-ABOUT-CLIENT contains 48 characters (15 + 1 + 20 + 1 + 2 + 3 + 3 + 3); the computer considers PERSONAL-INFO-ABOUT-CLIENT to be a string whose picture is X(48). If you say L PIC X(48), you can move all the PERSONAL-INFO-ABOUT-CLIENT to L by saying:

```
MOVE PERSONAL-INFO-ABOUT-CLIENT TO L.
```

To move the CLIENT-NAME to M, without moving the CLIENT-SEX, CLIENT-AGE, and WEIGHT-PROGRESS, say:

```
MOVE CLIENT-NAME TO M.
```

To write lots of information into a file, make the file's LINE be a group item.

How to extract from a file

Suppose you've already created a file whose name is EMPLOY; it's on disk or cards. Suppose the file contains information about employees. Suppose each line of the file contains 80 characters, as follows....

Characters 1-40: the employee's name
Characters 61-70: the employee's home phone number, including the area code
The other characters (41-60 and 71-80): "miscellaneous information", such as the employee's age, sex, address, salary, kind of job, and number of years with the company

Let's create a new file, called REPORT, on disk or on the printer's paper. Let's make REPORT contain just the employees' names and phone numbers, and omit the "miscellaneous information". Here's how:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
        PHONES.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
        The computer's name.
OBJECT-COMPUTER.
        The computer's name again.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT EMPLOY-FILE ASSIGN TO the location of EMPLOY.
        SELECT REPORT-FILE ASSIGN TO the location of REPORT.

DATA DIVISION.
FILE SECTION.
FD      EMPLOY-FILE the labeling for EMPLOY.
01      EMPLOY-LINE.
        02      EMPLOYEE-NAME PIC X(40).
        02      FILLER PIC X(20).
        02      HOME-PHONE.
                03      AREA-CODE PIC 999.
                03      PHONE-EXCHANGE PIC 999.
                03      REST-OF-PHONE-NUMBER PIC 9999.
        02      FILLER PIC X(10).
FD      REPORT-FILE the labeling for REPORT.
01      REPORT-LINE.
        02      EMPLOYEE-NAME-REPORTED PIC X(40).
        02      HOME-PHONE-REPORTED.
                03      LEFT-PARENTHESIS PIC X.
                03      AREA-CODE-REPORTED PIC 999.
                03      RIGHT-PARENTHESIS PIC X.
                03      PHONE-EXCHANGE-REPORTED PIC B999.
                03      THE-DASH PIC X.
                03      REST-OF-PHONE-NUMBER-REPORTED PIC 9999.

PROCEDURE DIVISION.
MAIN-ROUTINE-BEGINNING.
        OPEN INPUT EMPLOY-FILE.
        OPEN OUTPUT REPORT-FILE.
MAIN-ROUTINE-LOOP.
        READ EMPLOY-FILE AT END GO TO MAIN-ROUTINE-ENDING.
        MOVE EMPLOYEE-NAME TO EMPLOYEE-NAME-REPORTED.
        MOVE "(" TO LEFT-PARENTHESIS.
        MOVE AREA-CODE TO AREA-CODE-REPORTED.
        MOVE ")" TO RIGHT-PARENTHESIS.
        MOVE PHONE-EXCHANGE TO PHONE-EXCHANGE-REPORTED.
        MOVE "-" TO THE-DASH.
        MOVE PHONE-REMAINDER TO PHONE-REMAINDER-REPORTED.
        WRITE REPORT-LINE.
        GO TO MAIN-ROUTINE-LOOP.
MAIN-ROUTINE-ENDING.
        CLOSE EMPLOY-FILE REPORT-FILE.
        STOP RUN.
```

Throughout the DATA DIVISION, the special word "FILLER" stands for data the program won't mention later.

Characters 1-40 are EMPLOYEE-NAME.
Characters 41-60 are irrelevant.

Characters 61-63 are AREA-CODE.
Characters 64-66.
Characters 67-70.
Characters 71-80 are irrelevant.

The PICs say HOME-PHONE looks like this—
6172668128
but make HOME-PHONE-REPORTED look like this:
(617) 266-8128

The MAIN-ROUTINE-LOOP reads a line from EMPLOY-FILE, copies data into each part of REPORT-LINE, and then writes REPORT-LINE.

SORT

Suppose CUSTOM is a disk file that contains information about your customers. Suppose each line of the file contains 80 characters, as follows....

Characters 1-20: the customer's last name
Characters 21-80: other information about the customer

Here's how to put the file in alphabetical order, according to the customer's name:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
        ALPHA.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
        The computer's name.
OBJECT-COMPUTER.
        The computer's name again.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT CUSTOM-FILE ASSIGN TO the location of CUSTOM.
        SELECT SORT-FILE ASSIGN TO the location of SORT.

DATA DIVISION.
FILE SECTION.
FD      CUSTOM-FILE the labeling for CUSTOM.
01      CUSTOM-LINE PIC X(80).
SD      SORT-FILE.
01      SORT-LINE.
        02      LAST-NAME PIC X(20).
        02      FILLER PIC X(60).

PROCEDURE DIVISION.
MAIN-ROUTINE.
        SORT SORT-FILE
                ASCENDING KEY LAST-NAME
                USING CUSTOM-FILE
                GIVING CUSTOM-FILE.
        STOP RUN.
```

Putting a file in order, by alphabetizing or any other method, is called *sorting*. To sort the CUSTOM-FILE, the computer has to create a temporary disk file called a SORT-FILE.

In the sentence that says SELECT SORT-FILE, here's what to put for "the location of SORT":

Computer	The location of SORT
Eclipse	"SORT"
CDC	SORT
PDP-10, PDP-20	DSK DSK DSK RECORDING MODE ASCII
IBM using OS	UT-S-POEM
IBM using DOS	SYS001-UT-3330-S-SORTWK1

In the DATA DIVISION's FILE SECTION, the SD means a Sort-file Description. In the PROCEDURE DIVISION, the SORT sentence makes the computer automatically open the CUSTOM-FILE, sort it, and close it.

In the SORT sentence, if you replace ASCENDING by DESCENDING, the computer will sort the file in reverse order, so the Z's come first and the A's come last.

You can make the program fancier, by inserting extra statements before and after the SORT statement. But since the SORT statement automatically tells the computer to open CUSTOM-FILE, the CUSTOM-FILE must not be open already. If you already said OPEN CUSTOM-FILE, you must say CLOSE CUSTOM-FILE before you give the SORT statement.

If you replace GIVING CUSTOM-FILE by GIVING REPORT-FILE, the computer won't change CUSTOM-FILE, but will create a REPORT-FILE containing the information sorted. For REPORT-FILE, you must type an FD and SELECT it. The computer will automatically open it, so it must not be open already.

Within each line of CUSTOM-FILE, suppose characters 51-57 tell how much the customer bought from you during the past year. Let's find out which customers bought the most. Let's make the computer print the customer that bought the most, then the customer that bought the next most, etc. If two customers bought exactly the same amount, let's make the computer print their names in alphabetical order. This program does it:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
        BIGBUY.

The ENVIRONMENT DIVISION is the same as the previous program's.

DATA DIVISION.
FILE SECTION.
FD      CUSTOM-FILE the labeling for CUSTOM.
01      CUSTOM-LINE PIC X(80).
SD      SORT-FILE.
01      SORT-LINE.
        02      LAST-NAME PIC X(20).                    characters 1-20
        02      FILLER PIC X(30).                       characters 21-50
        02      AMOUNT-BOUGHT-DURING-YEAR PIC 99999V99. characters 51-57 (the V doesn't
        02      FILLER PIC X(23).                                          use a column)

PROCEDURE DIVISION.
MAIN-ROUTINE.
        SORT SORT-FILE
                DESCENDING KEY AMOUNT-BOUGHT-DURING-YEAR
                ASCENDING KEY LAST-NAME
                USING CUSTOM-FILE
                GIVING CUSTOM-FILE.
        STOP RUN.
```

The SORT sentence says: sort the file so that AMOUNT-BOUGHT-DURING-YEAR is in DESCENDING order; in case of a tie, put LAST-NAME is ASCENDING order.

MERGE

Suppose OLDCUS and NEWCUS are files (on disk or cards), as follows....

OLDCUS describes your old customers, and NEWCUS describes your newer customers. In those files, each line contains 80 characters; characters 1-20 contain the customer's last name. Each file is already in alphabetical order, by customer's last name.

Let's combine the two files. In other words, let's create a "combination" file (on disk or printer paper), called ALLCUS, that contains *all* the customers; and let's make ALLCUS be in alphabetical order also. Here's how:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
        MERGER.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
        The computer's name.
OBJECT-COMPUTER.
        The computer's name again.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT OLDCUS-FILE ASSIGN TO the location of OLDCUS.
        SELECT NEWCUS-FILE ASSIGN TO the location of NEWCUS.
        SELECT ALLCUS-FILE ASSIGN TO the location of ALLCUS.
        SELECT SORT-FILE ASSIGN TO the location of SORT.

DATA DIVISION.
FILE SECTION.
FD      OLDCUS-FILE the labeling for OLDCUS.
01      OLDCUS-LINE PIC X(80).
FD      NEWCUS-FILE the labeling for NEWCUS.
01      NEWCUS-LINE PIC X(80).
FD      ALLCUS-FILE the labeling for ALLCUS.
01      ALLCUS-LINE PIC X(80).
SD      SORT-FILE.
01      SORT-LINE.
        02      LAST-NAME PIC X(20).
        02      FILLER PIC X(60).

PROCEDURE DIVISION.
MAIN-ROUTINE.
        MERGE SORT-FILE
                ASCENDING KEY LAST-NAME
                USING OLDCUS-FILE NEWCUS-FILE
                GIVING ALLCUS-FILE.
        STOP RUN.
```

That program creates ALLCUS, which is a combination of OLDCUS and NEWCUS. To do that, the computer must create a SORT-FILE.

The word MERGE automatically opens and closes all the files involved. So do *not* say OPEN or CLOSE.

Warning: the word MERGE is in COBOL 74 but not in COBOL 68. So if your computer is old-fashioned and understands just COBOL 68, it doesn't understand the word MERGE.

If your computer understands the word MERGE, you can merge as many files as you like. For example, if you have files called CUS1, CUS2, CUS3, and CUS4, you can say:

```
MERGE SORT-FILE
        ASCENDING KEY LAST-NAME
        USING CUS1-FILE CUS2-FILE CUS3-FILE CUS4-FILE
        GIVING ALLCUS-FILE.
```

Before you MERGE, make sure that the files you're USING are already in alphabetical order.

SUBSCRIPTS

Like BASIC, FORTRAN, and PASCAL, COBOL lets you use subscripts.

For example, suppose your 4 favorite friends are SUE, JOE, TOM, and ANN. Let's make FAVORITE-FRIEND (1) be "SUE", FAVORITE-FRIEND (2) be "JOE", FAVORITE-FRIEND (3) be "TOM", and FAVORITE-FRIEND (4) be "ANN". Here's how:

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01      FAVORITE-FRIEND-TABLE.
        02      FAVORITE-FRIEND OCCURS 4 TIMES PIC XXX.  You have 4 FAVORITE-FRIENDs;
                                                         each has PIC XXX.
PROCEDURE DIVISION.
MAIN-ROUTINE.
        MOVE "SUE" TO FAVORITE-FRIEND (1).
        MOVE "JOE" TO FAVORITE-FRIEND (2).
        MOVE "TOM" TO FAVORITE-FRIEND (3).
        MOVE "ANN" TO FAVORITE-FRIEND (4).
        DISPLAY FAVORITE-FRIEND (1).
        DISPLAY FAVORITE-FRIEND (2).
        DISPLAY FAVORITE-FRIEND (3).
        DISPLAY FAVORITE-FRIEND (4).
        STOP RUN.
```

The computer will display:

```
SUE
JOE
TOM
ANN
```

When typing the program, remember to put a blank space before the subscript:

```
FAVORITE-FRIEND (1)
               ↑
          blank space
```

In COBOL, you say "OCCURS" instead of "DIMENSION":

BASIC	FORTRAN	PASCAL	COBOL
DIM F(4)	DIMENSION F(4)	F: ARRAY [1..4]	F OCCURS 4 TIMES

The subscript can be a variable. For example, instead of saying—

```
DISPLAY FAVORITE-FRIEND (1).
DISPLAY FAVORITE-FRIEND (2).
DISPLAY FAVORITE-FRIEND (3).
DISPLAY FAVORITE-FRIEND (4).
```

you can say:

```
DISPLAY FAVORITE-FRIEND (I).
```

To do that, you must tell the computer that the I goes from 1 to 4. Here's how:

```
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  01      FAVORITE-FRIEND-TABLE.
          02      FAVORITE-FRIEND OCCURS 4 TIMES PIC XXX.
→→01      I PIC 9.                                    I is a one-digit number.

  PROCEDURE DIVISION.
  MAIN-ROUTINE.
          MOVE "SUE" TO FAVORITE-FRIEND (1).
          MOVE "JOE" TO FAVORITE-FRIEND (2).
          MOVE "TOM" TO FAVORITE-FRIEND (3).
→→        PERFORM SHOW-FRIENDSHIP
→→                VARYING I FROM 1 BY 1 UNTIL I > 4.  I will be 1, 2, 3, 4.
          STOP RUN.
→→SHOW-FRIENDSHIP.
→→        DISPLAY FAVORITE-FRIEND (I).                I is the subscript.
```

To make the program run faster, say "COMP" at the end of the subscript's picture:

Computer	What to say
PDP, Eclipse	01 I PIC 9 COMP.
CDC	01 I PIC 9 COMP-1.
IBM	01 I PIC 9 COMP SYNC.

COMP stands for the word COMPUTATIONAL; SYNC stands for the word SYNCHRONIZED.

A subscript cannot contain an operation:

Okay	Wrong
FAVORITE-FRIEND (3)	FAVORITE-FRIEND (2 + 1)
	↑
	The + is not allowed.

Here's how to make Y-TABLE be a table that has 4 rows and 6 columns:

```
01      Y-TABLE.
        02      Y-ROW OCCURS 4 TIMES.
                03      Y OCCURS 6 TIMES PIC XXX.
```

The entire table is called:

Y-TABLE

The first row of Y-TABLE is called:

Y-ROW (1)

The second row of Y-TABLE is called:

Y-ROW (2)

The entry in the 2nd row and 5th column of Y-TABLE is called:

Y↑(2,↑5)

spaces

Suppose you teach 25 students, you've given each student 4 tests, and you want to store everything in a table. You want the table to contain 25 rows (a row for each student); in each row, you want the student's first name, middle initial, last name, and 4 scores. Here's how:

```
01      STUDENT-INFORMATION-TABLE.
        02      STUDENT-INFORMATION-ROW OCCURS 25 TIMES.
                03      FIRST-NAME PIC X(15).
                03      MIDDLE-INITIAL PIC X.
                03      LAST-NAME PIC X(20).
                03      TEST-SCORE OCCURS 4 TIMES PIC 999.
```

The entire table is called:

```
STUDENT-INFORMATION-TABLE
```

The table contains 25 rows; the first row is called:

```
STUDENT-INFORMATION-ROW (1)
```

The twelfth row is called:

```
STUDENT-INFORMATION-ROW (12)
```

The information in the twelfth row is called:

```
FIRST-NAME (12)
MIDDLE-INITIAL (12)
LAST-NAME (12)
TEST-SCORE (12, 1)
TEST-SCORE (12, 2)
TEST-SCORE (12, 3)
TEST-SCORE (12, 4)
```

Exotic features

IDENTIFICATION DIVISION

The IDENTIFICATION DIVISION can include these paragraphs:

```
PROGRAM-ID.
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.
```

In each paragraph after the PROGRAM-ID, put whatever garbage you please. The computer ignores everything the IDENTIFICATION DIVISION says.

The IDENTIFICATION DIVISION helps the computer center's librarian classify your program. The librarian wants the INSTALLATION paragraph to contain the computer center's name and address, the DATE-WRITTEN paragraph to tell when you finished debugging the program, the DATE-COMPILED paragraph to tell when the computer translated the program from COBOL into machine language, and the SECURITY paragraph to tell who may look at the program and who must not.

If you put the wrong date in the DATE-COMPILED paragraph, don't worry: when you ask the computer to produce a *COBOL listing* of your program, the listing will automatically show the correct date instead.

COMMENTS

The computer ignores any line that begins with an asterisk. So if you put this line in your program—

```
*THIS IS A LOUSY PROGRAM
```

— the computer will ignore the comment.

Create comments that explain how your program works. Put the comments near the bottom of the IDENTIFICATION DIVISION, near the top of the PROCEDURE DIVISION, and wherever your program looks confusing.

On PDP and Eclipse computers, put the asterisk at the far left; don't put any blank spaces before the asterisk. On IBM and CDC computers, put six blank spaces before the asterisk, so that the asterisk is in column 7.

DECIMAL-POINT IS COMMA

Some Europeans write commas instead of decimal points, and write decimal points instead of commas.

United States and England: 5,243,794.95
France and Italy: 5.243.794,95
Germany: 5 243 794,95

To write a COBOL program for a Frenchman, an Italian, or a German, make three changes....

Change #1. Insert this line:

```
DECIMAL-POINT IS COMMA.
```

Put that line in a SPECIAL-NAMES paragraph, at the end of the ENVIRONMENT DIVISION's CONFIGURATION SECTION:

```
CONFIGURATION SECTION.
SOURCE-COMPUTER.
       The computer's name.
OBJECT-COMPUTER.
       The computer's name again.
SPECIAL-NAMES.
       DECIMAL-POINT IS COMMA.
```

Change #2. Type all numbers in French-Italian-German notation. So instead of typing—

```
MOVE 5243794.95 TO K
```

type:

```
MOVE 5243794,95 TO K
```

Change #3. Use French-Italian-German notation in pictures for *edited* numbers.

For a Frenchman or an Italian: `K PIC Z.ZZZ.ZZZ,ZZ`
For a German: `K PIC ZBZZZBZZZ,ZZ`

Those are the only changes Europeans make. They still put a period at the end of every sentence, and still use English COBOL words such as MOVE and DISPLAY.

American COBOL: `DISPLAY "HELLO, STUPID".`
French COBOL: `DISPLAY "BONJOUR, BETE".`
German COBOL: `DISPLAY "GUTEN TAG, DUMMKOPF".`

Coupon

SHARE

Share your secret: photocopy this page, and give it to a friend. If you know a young (or young-at-heart) computerist who's having a birthday, Christmas party, or graduation, turn these volumes into a present: send me the computerist's name, address, and either a greeting card or what you want me to say.

VOLUMES

This tenth edition of *The Secret Guide to Computers* consists of eight volumes:

Volume	Topic
1: Popular BASIC	the easiest way to program, using many dialects of BASIC
2: Hassles in BASIC	advanced BASIC, data files, graphics, programming style
3: Popular Systems	kinds of computer equipment, how they work, who to buy from
4: Hassles in Systems	how to manage a computer center...and your career
5: Popular Applications	what computers have done and can do
6: Hassles in Applications	advanced programming techniques for advanced applications
7: Popular Languages	FORTRAN, COBOL, PASCAL, comparison with BASIC
8: Hassles in Languages	assembly languages, dozens of high-level languages

Each volume costs $3.70. The complete 8-volume set costs $29.60.

DISCOUNTS

To save money, get together with a friend:

20% discount if you order 2 complete sets (or any 16 booklets)
30% discount if you order 4 complete sets (or any 32 booklets)
40% discount if you order 10 complete sets (or any 80 booklets)
50% discount if you order 50 complete sets (or any 400 booklets)
60% discount if you order 500 complete sets (or any 4000 booklets)

POLICIES

Do *not* send extra money for postage and handling: I'll pay for them myself, even if you live in a foreign country. If you're not sure whether to order a book, go ahead: you can return unused books at any time, for a 100% refund.

CUT OUT THIS COUPON!

Circle the volumes you want: 1 2 3 4 5 6 7 8 ($3.70 each; $29.60 for all eight)

FREE brochures are available on the following topics; check off the ones you want:

___ how you can get intensive around-the-clock tutoring from Russ during the summer and pay practically nothing (we'll even give you housing)

___ how you can attend an intensive one-weekend course that Russ gives in Boston

___ how to make Russ give a short course in your own town, school, or company

___ how you can get permission to reprint *The Secret Guide to Computers*

___ how you can get later editions of *The Secret Guide to Computers*

___ how you can get *The Secret Guide to Tricky Living* (volume 1 is called "Sex"; volume 2 is called "Everything But")

On the other side, write your name, address, and any comments you have (pro or con) about *The Secret Guide to Computers*. Make check or money order payable to Russell M. Walter. Send to Russ Walter, 92 Saint Botolph St., Boston, MA 02116.

CONGRATULATIONS!

If you've read volumes 1, 2, 3, 4, 5, 6, and 7 of THE SECRET GUIDE TO COMPUTERS, you have just one more volume left! To get that volume 8— THE CLIMAX VOLUME— use the coupon on page 92. YOUR LIFE WILL BECOME A MANY-TONGUED ORGY.